Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

SpO2-Heartbeat Smartwatch Tracker: Monitorización de la saturación de oxígeno y la frecuencia cardiaca con un Smartwatch durante periodos de intensa actividad deportiva.

Alumno: Pablo Polo Mínguez

Tutor: José Vicente Álvarez.

Pablo Polo Mínguez

Índice

Tabla de contenido

Pablo Polo Mínguez

Agradecimientos

Me gustaría agradecer a todas las personas que me han ayudado con este proyecto.

Gracias a mi tutor, José Vicente Álvarez Bravo, por su apoyo, sus valiosas correcciones y su paciencia durante todo el desarrollo del proyecto.

Agradezco también a mi familia y amigos, cuyo ánimo y apoyo han contribuido para llegar hasta aquí y sobretodo su insistencia en que terminara el trabajo cada vez que posponía la fecha de presentación.

Gracias.

Pablo Polo Mínguez

Pablo Polo Mínguez

Resumen

Los dispositivos wearables son cada vez más comunes, relojes inteligentes, auriculares o gafas, ofreciendo nuevas posibilidades a aspectos de nuestra vida cotidiana, uno de los contextos más relevantes es el de la salud, gracias a estos dispositivos se pueden registrar parámetros biométricos y llevar a cabo un seguimiento de los mismos, así, el usuario no solo puede llevar un control más preciso de su bienestar, sino que también se abre la puerta a la detección temprana de posibles problemas de salud.

Este Trabajo de Fin de Grado se centra en el desarrollo de una ampliación en funcionalidades para la aplicación de relojes inteligentes con Wear OS del trabajo de fin de grado*: Wearable Heart Beat Register-Apps: Registro automatizado de la frecuencia cardiaca en reposo a través de un smartwatch* [54]

La aplicación existente se centraba en la monitorización de la frecuencia cardiaca en estado de reposo, mi proyecto busca la adaptación de todas las funcionalidades para incluir la monitorización de oxígeno en sangre del usuario y el registro de la frecuencia cardiaca en un estado de actividad y presentar todos los datos en una aplicación Android para Smartphone.

Abstract

Wearable devices are becoming increasingly common, such as smartwatches, headphones, and glasses, offering new possibilities for aspects of our daily lives. One of the most relevant contexts is health, thanks to these devices that can record biometric parameters and monitor them. This means that users can not only monitor their well-being more accurately, but also open the door to the early detection of potential health problems.

This end of degree project focuses on developing additional functionalities for the Wear OS smartwatch application from the final degree project: *Wearable Heart Beat Register-Apps: Registro automatizado de la frecuencia cardiaca en reposo a través de un smartwatch* [54]

The existing application focused on monitoring heart rate at rest. My project seeks to adapt all functionalities to include monitoring the user's blood oxygen levels and recording heart rate during activity, and to present all the data in an Android application for smartphones.

Parte I

Memoria del Proyecto

Pablo Polo Mínguez

Pablo Polo Mínguez

Capítulo 1

Descripción del proyecto

1.1. Introducción

En los últimos años, los dispositivos móviles han experimentado un auge en su popularidad, un tipo de dispositivos móviles en particular, los wearables, dispositivos que se tratan más como accesorios del propio teléfono, relojes, pulseras e incluso gafas o anillos que, además de cumplir con su función primaria, ofrecen una amplia variedad de funciones que facilitan la vida cotidiana, realizar llamadas, monitorizar el ritmo cardiaco o localizarnos mediante GPS entre otras funciones.

Los wearables más populares son los relojes inteligentes (smartwatch) y las pulseras de actividad (smartbands). Un sector que ha visto grandes beneficios de estos dispositivos es el ámbito de la salud. En España, más de 10 millones de personas sufren enfermedades relacionadas con el corazón, con más de 120.000 muertes anuales, según datos del INE.

Estos dispositivos permiten medir el ritmo cardíaco, la monitorización del sueño, la medición de los niveles de glucosa y la saturación de oxígeno. Además, muchos relojes inteligentes en el mercado incluyen algoritmos de inteligencia artificial que pueden detectar tempranamente enfermedades cardiovasculares o crisis epilépticas.

De este pretexto surge motivación para este trabajo de fin de grado, el objetivo es la ampliación de funcionalidades del proyecto existente: *Wearable Heart Beat Register-Apps: Registro automatizado de la frecuencia cardiaca en reposo a través de un smartwatch* [54]

Este proyecto tenía como objetivo el desarrollo de una aplicación para la monitorización de la frecuencia cardiaca en reposo de forma periódica en segundo plano, el objetivo de mi proyecto es la ampliación de funcionalidades de este trabajo para la lectura de saturación de oxígeno en sangre para esto se plantean 3 tipos de nuevos análisis a implementar, frecuencia cardiaca en estado de actividad deportiva intensa, análisis único de frecuencia cardiaca en reposo sin periodicidad y el análisis de la saturación de oxígeno en periodos de actividad deportiva intensa

Además de una aplicación Android para Smartphone que permitirá visualizar las mediciones realizadas con el reloj. A partir de estos registros, en el futuro se podría analizar los datos en busca de posibles anomalías, como la detección de arritmias, o enfermedades respiratorias como el asma.

1.2. Objetivos del trabajo

El objetivo del proyecto es extender la funcionalidad de la aplicación de Smart Watch existente, integrar el análisis de la saturación de oxígeno, controlar la actividad de forma activa y almacenar y visualizar dicha información en la app de su teléfono.

El desarrollo de una aplicación para dispositivos Smartwatch con sistema operativo Wear OS con las siguientes funcionalidades:

- Implementar la lectura de Oxigeno en sangre del usuario.
- Monitorizar la actividad de usuario para determinar que se encuentra en un periodo de actividad física.
- Transmitir los valores registrados a la aplicación móvil, mediante conexión bluetooth en una base de datos.

El desarrollo de una aplicación móvil en Android con las siguientes funcionalidades:

- Determinar valores recomendados para cada usuario basándonos en su edad, peso y género.
- Visualizar los datos mediante gráficos y organizarlos cronológicamente y por su ubicación con un mapa.

1.3 Estado del arte.

En este apartado se analizarán las circunstancias actuales en la que se encuentran los diferentes dispositivos, aplicaciones y sistemas operativos similares al proyecto, relojes inteligentes de diferentes marcas, con sus propias aplicaciones de salud.

En el contexto de la monitorización de frecuencias cardiacas y saturación de oxigeno no vamos a comparar las aplicaciones desarrolladas durante el proyecto con las de empresas multimillonarias con miles de desarrolladoras se busca más mencionar las opciones disponibles y con qué características cumple mi proyecto.

De las opciones disponibles en el mercado se van a tener en cuenta las 3 más populares, si tienen un sensor SpO2, el responsable de la lectura de saturación de oxígeno, si el uso de los metadatos es público, es decir se podría acceder a ellos desde una aplicación o proyecto externo como la que este trabajo utiliza para la aplicación de Smartphone.

Modelo (SO)	Frecuencia Cardiaca continua	SpO_2	APIs	Metadatos accesibles	Notas
Apple Watch Series 10 (watchOS)	✔	✔	HealthKit	✖	Alta adopción; restricción/regulación específica en EE. UU. a documentar. [48] [49]
Samsung Galaxy Watch7 (Wear OS)	✔	✔	Samsung Health ⇄ Health Connect (Android)	✖	Buen encaje con Android/Wear OS; sincronización no garantizada en tiempo real. [50][51]
HUAWEI Watch GT 4 (HarmonyOS)	✔	✔	Huawei Health Kit (HMS Core)	✔	Muy extendido en Europa; integración viable vía HMS/Health. [52][53]
Mi aplicación (Wear OS)	✔	✔	Android Sensor SDK	✔	

1.4 Entorno y tecnologías utilizadas

Se detallan las tecnologías y entornos usados en el proyecto, los sistemas operativos, Android y Wear OS, y el lenguaje de programación java.

1.3.1 Android

Android es un sistema operativo móvil basado en una versión adaptada del kernel de Linux y otros componentes de software de código abierto. Está diseñado principalmente para dispositivos móviles con pantalla táctil, como smartphones y tablets. Desarrollado originalmente por un grupo de desarrolladores conocido como Open Handset Alliance, aunque la versión que conocemos en la actualidad y existe en la mayoría de los teléfonos móviles es de Google.

El sistema operativo en su forma básica se conoce como el Proyecto de Código Abierto de Android (AOSP) y es un software gratuito y de código abierto (FOSS), distribuido principalmente bajo la Licencia Apache. No obstante, la mayoría de los dispositivos utilizan una versión propietaria de Android desarrollada por Google, que incluye software adicional de código cerrado preinstalado. Entre estos se encuentran los Google Mobile Services (GMS), que incluyen aplicaciones esenciales como Google Chrome, la tienda digital Google Play y la plataforma de desarrollo Google Play Services.

El código fuente de Android ha sido adaptado para desarrollar variantes específicas del sistema operativo en diversos dispositivos electrónicos, como consolas de videojuegos, cámaras digitales, reproductores multimedia portátiles y ordenadores personales, cada uno con una interfaz de usuario personalizada. Entre los derivados más conocidos se encuentran Android TV, diseñado para televisores, y Wear OS, enfocado en dispositivos portátiles, ambos desarrollados por Google. En este proyecto, se ha utilizado Wear OS, cuya implementación se explicará más adelante.

Android se ha consolidado como el sistema operativo más vendido a nivel mundial en teléfonos inteligentes desde 2011 y en tabletas desde 2013. Para mayo de 2021, contaba con más de tres mil millones de usuarios activos mensuales. Además, en enero de 2021, Google Play Store ofrecía más de tres millones de aplicaciones.

La arquitectura de Android se organiza en capas dispuestas de arriba hacia abajo, explicadas a continuación.

Aplicaciones: Las aplicaciones básicas de Android incluyen herramientas esenciales como correo electrónico, mensajería de texto, calendario, mapas, navegador, contactos y muchas más. Estas aplicaciones están desarrolladas en el lenguaje de programación Java, el cual se detallará más adelante, ya que también es el lenguaje utilizado en este proyecto.

Framework o de las aplicaciones: Consisten en un conjunto de herramientas de software que permiten a los desarrolladores crear productos finales que satisfacen los requisitos establecidos por los propietarios del proyecto.

Bibliotecas: Android incorpora una serie de bibliotecas escritas en C/C++, utilizadas por diversos componentes del sistema. Estas bibliotecas son accesibles para los desarrolladores a través del framework de aplicaciones de Android. Por ejemplo, en este proyecto se ha empleado SQLite, cuya implementación será descrita con mayor detalle en el capítulo correspondiente.

Runtime de Android: Es el entorno de ejecución para aplicaciones en Android. ART (Android Runtime) reemplazó a Dalvik, la máquina virtual originalmente utilizada por Android, y se encarga de transformar las aplicaciones en instrucciones de máquina que son ejecutadas por el entorno nativo del dispositivo.

Kernel de Linux: Android se basa en el kernel de Linux para ofrecer servicios fundamentales del sistema, como seguridad, gestión de memoria, administración de procesos, conectividad en red y controladores de hardware. Este núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

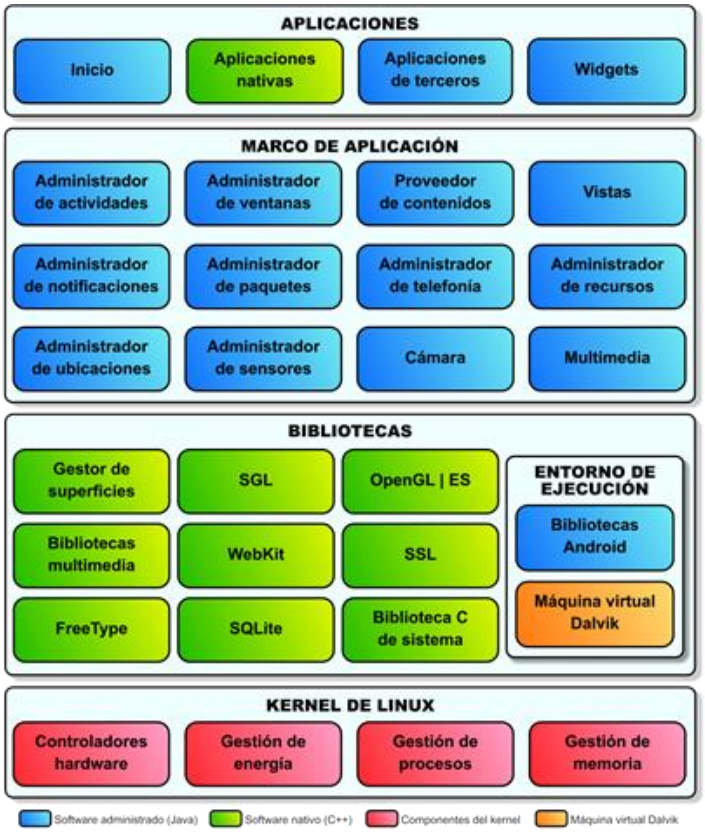


Figura 1.1: Arquitectura Android [44]

1.3.2 Wear OS

Wear OS es una versión del sistema operativo de Android diseñada específicamente para relojes inteligentes y otros Wearables, se enfoca en optimizar las funcionales específicas que benefician el uso de este tipo de dispositivos, es vinculable con dispositivos con Android 6.0 *"Marshmallow"* o versiones posteriores del sistema operativo, también dispone de un soporte limitado en dispositivos IOS a partir de la versión 10.0

Wear OS admite conectividad mediante Bluetooth, NFC, Wi-Fi, 3G y LTE, además de ofrecer una amplia variedad de funciones y aplicaciones. Los relojes que utilizan este sistema operativo pueden tener pantallas de forma cuadrada, circular o rectangular. Entre las marcas que fabrican dispositivos con Wear OS se encuentran Samsung, Oppo y Fossil.

En cuanto a su arquitectura, aunque comparte la base con Android, presenta ciertas diferencias importantes. Wear OS está diseñado para funcionar en dispositivos con recursos más limitados que los teléfonos móviles, por lo que su arquitectura está optimizada para trabajar bajo estas restricciones de hardware. Además, depende en mayor medida de la comunicación con un smartphone emparejado para acceder a datos y servicios en línea, lo que refuerza su integración con dispositivos móviles.

Otra diferencia clave es que Wear OS está diseñado para dispositivos pequeños y portátiles, por lo que no requiere elementos como pantallas grandes, cámaras o puertos de entrada/salida que son comunes en los smartphones. Estas diferencias hacen que la arquitectura de Wear OS se adapte específicamente a las necesidades de los dispositivos wearables, ajustándose tanto en funcionalidad como en requerimientos técnicos.

1.3.3. Java

Java es un lenguaje de programación de alto nivel, orientado a objetos y basado en clases, diseñado para minimizar las dependencias de implementación, es el lenguaje oficial para el desarrollo de aplicaciones en Android. Una característica clave de Java para el desarrollo de aplicaciones Android es su enfoque en el modelo de programación orientada a eventos, los eventos pueden ser llamados tanto por el usuario como por el sistema operativo.

Estos eventos son gestionados por los diferentes componentes de la aplicación, activities, fragments, servicios y broadcast receivers, que son los elementos fundamentales de una aplicación Android, a lo largo de la memoria se verán en más detalle el uso de los componentes Java del proyecto, en vez de profundizar en este apartado.

1.5. Estructura de la memoria

La memoria del proyecto se divide en tres secciones principales, cada una compuesta por varios capítulos.

Parte I: Memoria del proyecto

- **Introducción**: Se presenta una introducción al tema del Trabajo de Fin de Grado, junto con los objetivos del proyecto. además, se compara el proyecto con otros dos trabajos similares que también desarrollaron aplicaciones para dispositivos wearables.
- **Metodología**: Este capítulo describe el proceso de desarrollo, detallando la metodología adoptada y las herramientas empleadas durante el proyecto.
- **Planificación**: Se incluyen las estimaciones de tiempo y presupuesto requeridos para el proyecto. Contiene una planificación inicial realizada antes del desarrollo, una planificación final que compara los esfuerzos reales con las estimaciones previas, y el presupuesto utilizado.

Parte II: Documentación técnica

- **Análisis**: Se centra en la ingeniería de requisitos, los requisitos del proyecto, funcionales y no funcionales, diagramas de casos de uso, historias de usuario derivadas, el modelo de dominio y diagramas de actividad.
- **Diseño**: Este capítulo aborda la arquitectura y los patrones de diseño empleados en las aplicaciones desarrolladas. También incluye diagramas como el de la base de datos, diagramas de paquetes y de despliegue, además de describir la estructura final del proyecto.
- **Implementación**: Resume el proceso de implementación de ambas aplicaciones. Se explican aspectos como la detección del reposo, las opciones disponibles para monitorizar la ubicación y los acelerómetros, las opciones para la lectura del oxígeno en sangre y la solución elegida finalmente.

Parte III: Manuales de la aplicación

- **Manual de Instalación**: Describe paso a paso cómo instalar ambas aplicaciones desde cero. Incluye la instalación de Android Studio, la clonación del proyecto desde GitHub y la activación de la depuración en los dispositivos. Todo se explica de manera detallada.
- **Manual de Usuario**: Instrucciones sobre cómo usar las aplicaciones una vez instaladas, explicando sus funcionalidades y características principales.

Finalmente, la memoria concluye con la bibliografía utilizada a lo largo del desarrollo del proyecto.

Capítulo 2

Metodología

2.1. Proceso de desarrollo

En este apartado se explican brevemente las metodologías planteadas, como se comparan entre si cual se eligió finalmente y porque, se valoró en especial la facilidad para el desarrollo que conlleva cada metodología, flexibilidad de cara a la planificación y diseño antes de empezar a codificar y la posibilidad de corregir errores además de la familiarización personal existente con la metodología Agile al haberse utilizado en proyectos pasados, se comparan el modelo en cascada y modelo ágil, se eligió una metodología ágil de iteraciones o Sprint.

La metodología en cascada es el planteamiento más tradicional, en algunos casos se considera anticuado, consiste en separar el desarrollo en fases, se realiza todo el análisis y diseño, la parte teórica y luego se programa la aplicación entera, presenta problemas a la hora de modificar, partes del desarrollo, que es algo que casi siempre pasa en estos proyectos, mientras que la metodología ágil se basa en iteraciones, teniendo cada iteración objetivos aislados, lo que permite avanzar de forma progresiva, ir viendo cómo avanza la aplicación de forma práctica y facilitando el volver a una iteración pasada para corregir o modificar el desarrollo.

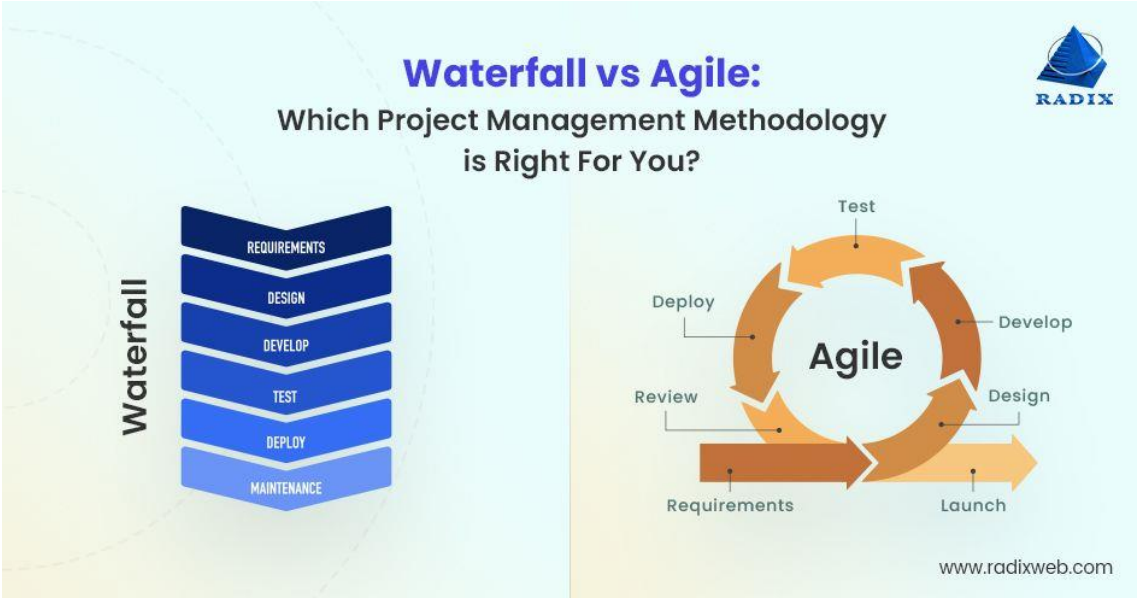


Figura 2.1 Agile vs waterfall [43]

Después de comparar las dos posibilidades se optó por una metodológica Agile, ya que presenta una mayor flexibilidad para el desarrollo y las entregas parciales facilitan la visualización del progreso, lo que hace que se puedan ir implementando nuevas mejoras que en un principio no estaban planeadas y corregir problemas a medida que van surgiendo y no tener que afrontarlos todos en la etapa final del proyecto.

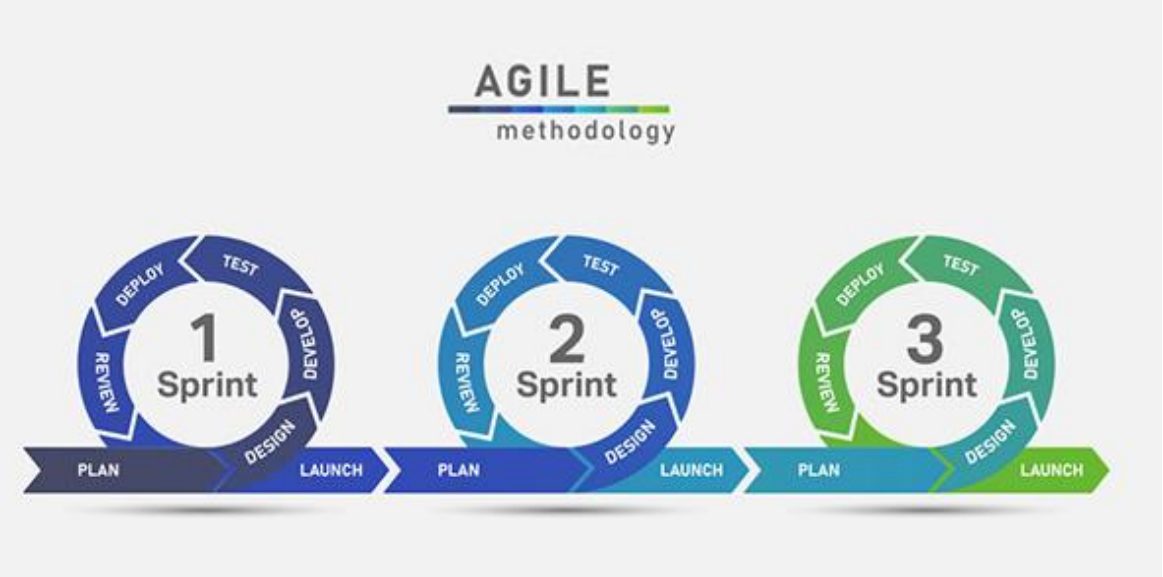


Figura 2.2 Metodología Agile [55]

2.2. Herramientas utilizadas

Esta sección detalla las herramientas usadas durante el desarrollo del proyecto, tanto hardware como software.

Hardware

- El proyecto se desarrolló principalmente en un ordenador personal de torre, con sistema operativo Windows 10 y especificaciones de hardware de gama media-alta.
- Teléfono móvil Xiaomi Pocophone F3 con sistema operativo Android para las pruebas y debugging de la aplicación móvil
- Smartwatch proporcionado por la universidad, Galaxy Watch 4, para las pruebas y optimización de la aplicación del reloj, el debugging de la aplicación del reloj se realizó desde el emulador de Android studio ya que facilitaba mucho el realizar cambios en el código sin tener que actualizar la aplicación en el Smart Watch.

IDE

- Se planteó brevemente el uso del entorno de desarrollo de Jetbrains, IntelliJ Idea, ya que se había trabajado con su versión para Python, PyCharm extensivamente, en un entorno de trabajo profesional, finalmente se optó por Android studio ya que el IDE está más enfocado al desarrollo de aplicaciones Wear OS.

Documentación

- La memoria se ha realizado con el procesador de texto de Windows, Word al ser la opción más cómoda y estándar, se planteó utilizar LaTeX con el editor online de overleaf, que simplifica su uso, y con la que se ha trabajado anteriormente, pero se concluyó que para una memoria tan extensa no merecía la pena ya que el producto final al ser un pdf no presenta diferencia de formato.
- Los diagramas se han hecho con draw.io [1], una herramienta online de diseño UML.

Control de versiones.

- El control de versiones se ha realizado con la extensión de GitHub de Android Studio, directamente sobre el mismo repositorio, muy útil para seguir trabajando desde cualquier otro ordenador o para volver a una versión anterior cuando se comete un error o comprobar los cambios de forma precisa con un historial.

//todo metodología de iteración o sprint

Capítulo 3

Planificación

3.1. Planificación temporal

Este apartado recoge la previsión de cuanto se tardará en realizar el proyecto, dentro de esta sección se detalla una planificación inicial de lo que se estima que se tardará en completar el desarrollo que posteriormente se comparara con lo que se tardó en realidad en la planificación final.

3.1.1. Planificación inicial

En esta sección se describe la planificación inicial del TFG, diseñada para cumplir con los objetivos establecidos y satisfacer todos los requisitos dentro del plazo definido. Se va a emplear una metodología ágil basada en iteraciones o incrementos, los cuales irán incorporando nuevas funcionalidades hasta completar el proyecto.

A continuación, se detalla la distribución de las horas asignadas a cada iteración, junto con un análisis de los posibles riesgos que podrían retrasar el desarrollo del proyecto. Esta planificación horaria se ha realizado teniendo en cuenta mi experiencia pasada con el desarrollo de prácticas durante la carrera y mi experiencia programando en un entorno de trabajo. Una vez distribuidas las funcionalidades previstas para cada iteración, se ha estimado su duración en horas basada en dicha experiencia previa.

La planificación del proyecto comienza a partir de junio de 2024, debido a la limitación temporal de estar trabajando mientras se realiza el proyecto se estima que unas medias de 2 horas diarias van a ser dedicadas al proyecto.

Teniendo en cuenta la normativa de 25 horas por cada crédito E.T.C.S y el trabajo de fin de grado constituye 12 créditos se estiman 300 horas de trabajo o 150 días o 5 meses, se plantea una planificación más extensa de lo habitual para un trabajo de fin de grado

1º Iteración (80 horas): La primera iteración tiene como objetivo familiarizarse con el entorno de programación, entender el funcionamiento de la aplicación existente y la creación de una pequeña aplicación mock-up con uso limitado para asimilar el uso de los sensores de aceleración, frecuencia cardiaca y localización.

2º Iteración (100 horas): La segunda iteración se centra en el uso de hilos, interacción entre los componentes de la aplicación, y la implementación de las funcionalidades para la lectura de saturación de oxígeno, la iteración se considera finalizada cuando se tiene una versión funcional que cumple los requisitos funcionales establecidos.

3º Iteración (80 horas): Una vez tenemos una aplicación de smartwach capaz de realizar análisis, el desarrollo se centra en la aplicación del dispositivo móvil, el objetivo de esta iteración es transmitir y almacenar los datos a la app móvil, adaptar la base de datos para los mismos y la visualización de un historial de gráficos mediante consultas a la base de datos.

4º Iteración (40 horas): La ultima interacción está reservada para análisis y comparaciones de las lecturas del oxígeno con aplicaciones existentes, considerando posibles modificaciones a la formula y dejando tiempo para correcciones y contratiempos, además de volver a iteraciones anteriores para incluir mejoras o posibles cambios considerados a posteriori, en resumen, enfocada a depurar lo desarrollado en iteraciones anteriores.

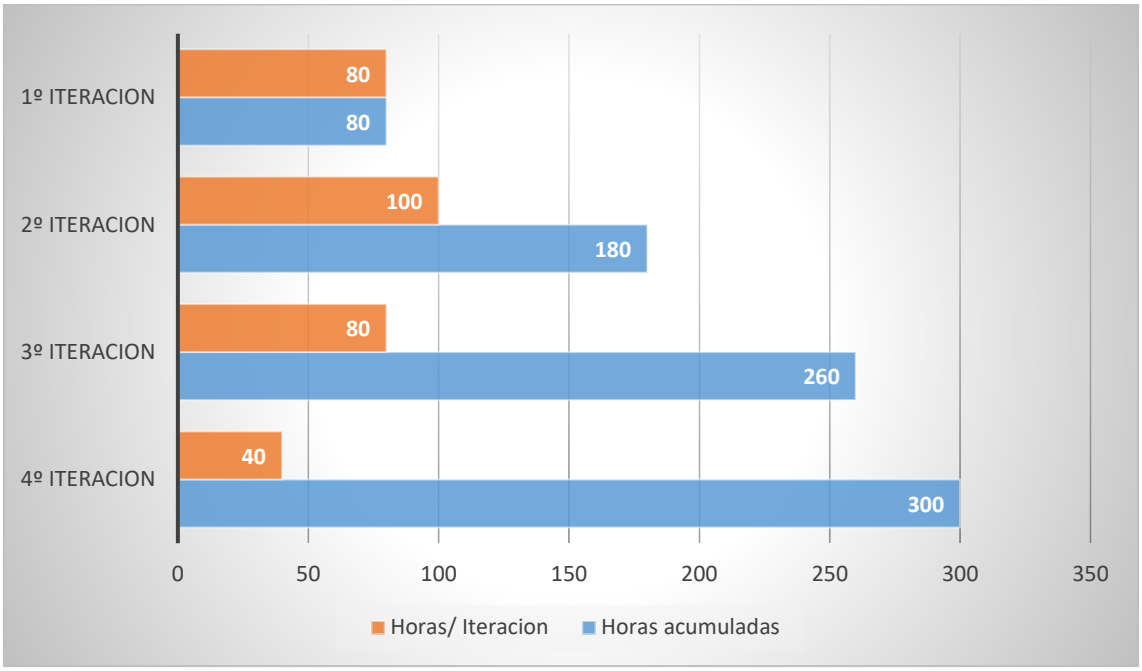


Figura 3.1 Distribución temporal inicial por iteraciones

3.2.2. Análisis de riesgos

El análisis de riesgos tiene en cuenta posibles contratiempos que se pueden experimentar a lo largo del proyecto y cómo van a poder afectar al tiempo de desarrollo, se plantean los siguientes riesgos con su probabilidad y la pérdida de tiempo que pienso que supondrían en la tabla 3.2.

ID	Riesgo	Probabilidad	Tiempo
1	Salud	0.10	25
2	Falta de experiencia en desarrollo Android	0.2	25
3	Falta de experiencia en desarrollo Wear OS	0.4	20
4	Planificación incorrecta	0.5	15
5	Problemas con el Hardware	0.05	10
6	Perdida de datos/progreso	0.01	20

Figura 3.2 Tabla de riesgos.

El tiempo que se usa para calcular como podrían afectar estos riesgos al proyecto se denomina exposición, se calcula multiplicando la probabilidad de que un riesgo ocurra por su tiempo en horas, detallado en la tabla 3.3.

ID	Riesgo	Exposición	Plan de reducción	Plan de mitigación
1	Salud	2.5	Evitar el riesgo de contagio o enfermedad	Ir al médico o intentar recuperarme lo mas rápido posible
2	Falta de experiencia en desarrollo Android	5	Leer documentación oficial, ver tutoriales	Dedicarle más tiempo o pedir ayuda al tutor
3	Falta de experiencia en desarrollo Wear OS	8	Leer documentación oficial, ver tutoriales	Dedicarle más tiempo o pedir ayuda al tutor
4	Planificación incorrecta	7.5	Planificar con más tiempo de margen	Dedicarle más tiempo diario al proyecto
5	Problemas con el Hardware	0.5	Cuidar bien el ordenador y el reloj	Arreglar o comprar un dispositivo nuevo o usar otro ordenador.
6	Perdida de datos/progreso	0.2	Con el uso de git es casi imposible que se dé esta posibilidad	Dedicar más horas para compensar el progreso perdido

Figura 3.3 Exposición de los riesgos

3.1.3. Planificación final

Una vez finalizado casi por completo el proyecto, resulta imprescindible analizar cómo se ha desarrollado realmente y comparar este progreso con las estimaciones iniciales. Este análisis permitirá evaluar en qué medida la planificación se ajustó a la realidad.

En esta sección se detallan las iteraciones planificadas al inicio del proyecto, explicando las tareas realizadas en cada una de ellas. Además, si se observan diferencias significativas entre los tiempos estimados y los tiempos reales, se ofrecerá una justificación basada en los riesgos identificados previamente, esta diferencia se visualiza en la gráfica 3.4.

El tiempo real se ha cuantificado utilizando los comits de github, que se realizaron al finalizar cada iteración.

1º Iteración: Esta iteración tenía como objetivo la familiarización con el entorno, la Api de Android, la programación Wear OS, para esto se desarrolla una app con funcionalidades limitadas, el uso de los sensores de frecuencia y ubicación.

El contacto inicial con el tipo de programación fue un poco menos intuitivo de lo anticipado, y se tardó bastante en empezar a programar como tal, dedicando las 3 primeras semanas a ver tutoriales, leer el código de la app existente y probarla con la herramienta de debbug para acostumbrarme al entorno, a partir de ahí empecé a programar la aplicación de prueba, se consiguió cumplir con los objetivos en el tiempo estimado de 80 horas o 6 semanas.

2º Iteración: La segunda iteración se enfocaba en la integración de todas las funcionalidades nuevas para la app del smartwatch, se estimó que iba a ser la iteración más larga con 100 horas.

Se comenzó con un planteamiento de la lectura de saturación de oxigeno erróneo, el uso del sensor de la SDK de Samsung, comentado más en detalle en [], se invierten unas 14 horas intentando utilizar este sensor, intentando incluso inscribirme en el programa de Partners de Samsung, pero finalmente se descarta la opción, pese a este contratiempo, las 100 horas estimadas inicialmente resultaron ser demasiado tiempo ya que la versión desarrollada que cumplía los requisitos y cerraba la iteración se completó a las 7 semanas (98 horas) incluyendo las dos semanas perdidas.

3º Iteración: Ya con una versión de la app del reloj capaz de generar datos, el objetivo de esta iteración es el desarrollo de la app móvil, transmitir y almacenar los datos además de mostrarlos.

La iteración termino antes de lo estimado, el desarrollo en Android para la app de móvil me resulto más cómodo además de que la complejidad era menor, la iteración se completó en 56 horas, en comparación a las 80 horas estimadas.

4º Iteración: La mayor parte del tiempo se empleó para la comparativa de resultados y planteamientos en el cambio de fórmulas junto con pruebas para asegurarse del correcto funcionamiento, sobre todo a la hora de cargar datos diferentes a la base de datos para realizar pruebas, unas 30 horas, el resto del tiempo se invirtió en los cambios que surgieron, entre ellos la inclusión de una barra de progreso y un cambio en la visualización de las gráficas, además de una ligera modificación en el funcionamiento de la app del reloj, en total 56 horas comparadas con las 40 estimadas, algunas iteraciones llevaron menos tiempo, como la tercera y otras un poco más como la última, en general el tiempo no supero al total estimado aun con contratiempos.

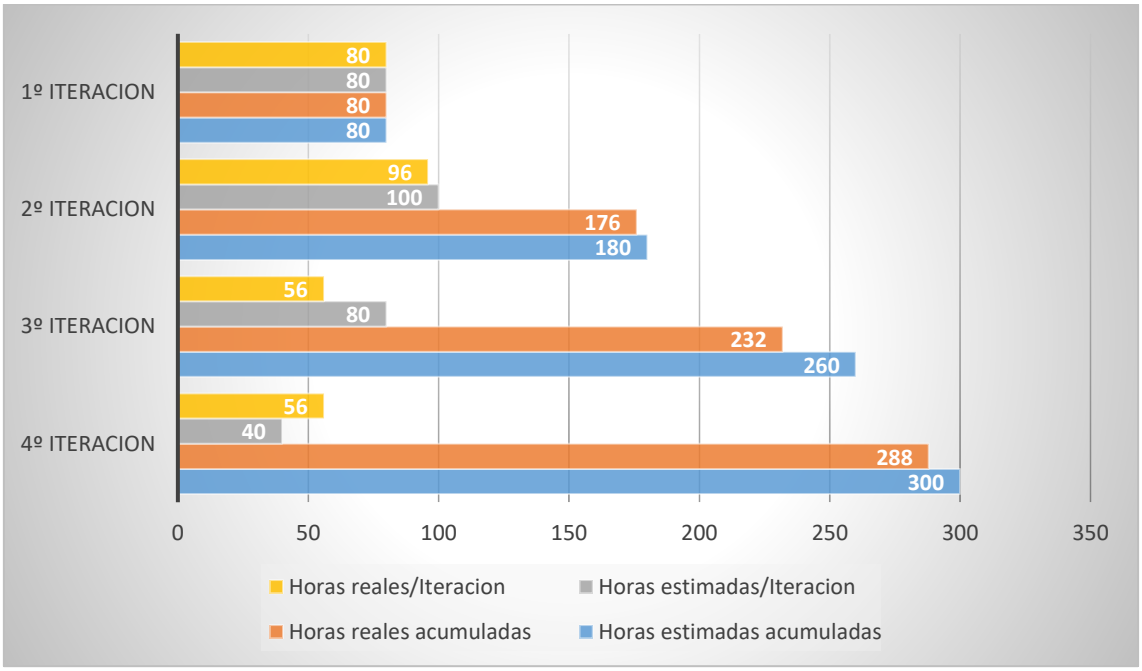


Figura 3.4 Distribución temporal inicial y final por iteraciones

3.2. Presupuesto económico

En este apartado se estima un posible presupuesto hipotético para el proyecto, evidentemente la cuantificación del coste de los recursos humanos o el hardware utilizado no va a ser realista ya que se han trabajado 2 horas diarias y el hardware es un ordenador personal que tiene 10 años, pero se van a realizar suposiciones como si los dispositivos fuesen nuevos y el sueldo fuese un sueldo estándar a tiempo parcial.

3.2.1. Hardware y software.

En cuanto al software, todos los programas empleados han sido gratuitos, por lo que no se incluye ningún gasto en este apartado.

Para calcular los costes amortizados asociados a los dispositivos, se estima el uso basado en el tiempo efectivo de trabajo durante el proyecto en comparación con su tiempo de vida, este enfoque es más justo que considerar el coste total de los dispositivos. A partir del precio de compra de cada dispositivo, se calculó su coste proporcional en función de su vida útil promedio.

La fórmula que se usa para calcular el coste amortizado es:

$$\frac{\text{horas de uso en el proyecto}}{horas\ de\ vida\ util}\ x\ precio$$

Ordenador: Adquirido por 1500 euros, con una vida útil estimada de 10 años. Para las 300 horas de trabajo previstas en este TFG, el coste amortizado del ordenador se calcularía como $\frac{300}{87600}\ x\ 1500$ = 5.13 euros

Smartphone: Comprado por 200 euros, con una vida útil de 4 años.

Smartwatch: Adquirido por 156 euros, con una vida útil estimada de 3 años.

Dispositivo	Coste amortizado	Precio adquisición
Ordenador	5.13 €	1500 €
Smartphone Android	2.07 €	200 €
Smartwatch Samsung	2.15 €	156 €
Total	7.35€	1856€

Figura 3.5 Presupuesto Del Hardware

3.2.2. Recursos humanos

Como se comentaba en la introducción de la sección se considera un sueldo a tiempo parcial durante 6 meses, si la duración del proyecto es de 300 horas, se van a considerar 14 horas semanales, el sueldo medio de un programador Android en España [18] y el salario a tiempo parcial seria aproximadamente la mitad, así que se plantea el salario hipotético de 700 € al mes, el coste de los recursos humanos seria 4200€.

3.2.3. Presupuesto total

En esta sección se calcula el coste total estimado del proyecto, considerando los gastos relacionados con el hardware, el software y los recursos humanos, ya analizados previamente. No obstante, también se incluyen otros gastos que son igualmente relevantes, como los asociados al espacio de trabajo.

El espacio de trabajo abarca gastos comunes a la mayoría de los proyectos, como el consumo de electricidad y la conexión a internet. Aunque el coste del lugar de residencia podría ser incluido, en este caso no se aplica, ya que el proyecto se desarrolló en mi residencia habitual.

La luz se calcula utilizando la media del coste en España durante los meses del desarrollo del proyecto, 0.1465 €/kwh según los datos de la siguiente página [19], siendo el tiempo de desarrollo 300 horas el coste total es de 43.95 € el precio del internet es de 30 € al mes, no se debería considerar el coste íntegro para el proyecto ya que es internet que para uso personal pero para simplificar los cálculos se asume como si fuese internet de una oficina o de un lugar de trabajo y se añade directamente.

Gasto	Coste
Software y Hardware	7.35 €
Recursos Humanos	4200 €
Luz	43.95 €
Internet	120 €
Total	4268.55 €

Figura 3.6 Presupuesto Total

Parte II

Documentación técnica

Pablo Polo Mínguez

Capítulo 4

Análisis

En esta sección se presenta el análisis del sistema desarrollado, partiendo de la identificación de los requisitos y la definición de las funcionalidades clave que debe cumplir, las historias de usuario, que permiten comprender las necesidades del usuario desde una perspectiva funcional y el modelo de dominio donde se representan los elementos clave del sistema y cómo se relacionan entre sí, finalmente, se incluyen los diagramas de actividad, que explican paso a paso el flujo de los procesos principales.

4.1. Requisitos

En este apartado se definen los requisitos del proyecto. Estos definirán los criterios que el sistema debe cumplir para garantizar su funcionamiento y el cumplimiento de los objetivos planteados, se van a clasificar en 3 grupos, requisitos funcionales, no funcionales y requisitos de información.

Los requisitos funcionales describen las características y funcionalidades específicas que debe ofrecer el sistema para cumplir los requerimientos del usuario por ello se incluyen los requisitos de usuario en este apartado.

Si los requisitos funcionales se refieren a lo que hace el sistema los no funcionales describen como lo hace, se van a centra en aspectos como el rendimiento, la seguridad y la usabilidad,los requisitos de información describirán los datos del sistema.

4.1.1. Iteración 1

La primera iteración de desarrollo tendrá como objetivo familiarizarse con el contexto de trabajo de desarrollo java en Android y Wear OS, la mayor parte de esta iteración consistirá en documentarse ver tutoriales e implementar una versión cruda de la aplicación que trabaje con interfaces, funcionalidades y datos simples.

ID	Nombre	Descripción
RF_01	Inicio del análisis en reposo	El sistema debe permitir al usuario comenzar el análisis en reposo.
RF_02	Inicio del análisis activo	El sistema debe permitir al usuario comenzar el análisis de forma activa.
RF_03	Lectura de ubicación	El sistema debe almacenar y mostrar la ubicación del reloj
RF_04	Lectura de aceleración	El sistema debe almacenar y mostrar la aceleración del sensor durante el análisis.
RF_05	Lectura de frecuencia cardiaca	El sistema debe almacenar y mostrar la frecuencia cardiaca del sensor en cada momento del análisis.

Figura 4.1 Requisitos Funcionales 1º Iteración

ID	Nombre	Descripción
RNF_01	Entorno desarrollo	El sistema debe desarrollarse de forma compatible con Wear OS utilizando el lenguaje Java.
RNF_02	Ahorro de batería	El sistema debe llevar a cabo los análisis de forma que priorice el bajo consumo de batería.

Figura 4.2 Requisitos No Funcionales 1º Iteración

4.1.2. Iteración 2

En la segunda iteración se adaptará completamente la app del smartwatch para incluir todas las funcionalidades relevantes para llevar a cabo el análisis de la saturación de oxígeno y almacenar los datos para en la siguiente iteración almacenarlos en la base de datos.

ID	Nombre	Descripción
RF_06	Navegación del menú inicial	El sistema debe permitir al usuario desplazarse por la app para elegir el análisis que quiera
RF_07	"Perfil" usuario	El sistema debe pedir y almacenar el peso, género y edad del usuario.
RF_08	Monitorización actividad	El sistema deberá determinar si el usuario se encuentra en reposo o en estado activo dependiendo del análisis.
RF_09	Interrumpir análisis	El sistema debe interrumpir el análisis si el usuario no se encuentra o sale del estado necesario para dicho análisis ya sea en reposo o de forma activa.
RF_10	Completar análisis	El sistema debe ser capaz de finalizar los análisis de forma satisfactoria ya sea el análisis activo o en reposo
RF_11	Mostrar estado	El sistema debe mostrar el estado en que se encuentra el usuario reposo o en movimiento.

Figura 4.3 Requisitos Funcionales 2º Iteración

ID	Nombre	Descripción
RNF_03	Condiciones inicio del análisis activo	Las condiciones para que se realice el análisis de forma activa serán que los acelerómetros estén por encima de 3 m/s.
RNF_04	Condiciones Interrupción del análisis activo	El análisis activo se interrumpirá si se incumplen las condiciones de RNF_03, que los acelerómetros bajen de 3 m/s.
RNF_05	Duración del análisis activo	El análisis activo tendrá una duración de 30 segundos.
RNF_06	Margen para entrar en reposo	El sistema deberá dejar unos 2 segundos de margen durante la monitorización activa antes de que se considere el estado de reposo.

Figura 4.4 Requisitos No Funcionales 2º Iteración

4.1.3. Iteración 3

En esta última iteración se tratarán los datos analizados previamente y se almacenarán en la base de datos además de modificar la interfaz de la aplicación de móvil y las consultas realizadas en la base de datos junto con la forma de tratar los datos obtenidos por las consultas de cara a mostrarlos en la interfaz.

Inicialmente la iteración 3 estaba dividida en una 3º y 4º iteración, pero decidí juntarlas ya que la primera parte me llevo menos tiempo del esperado y al final acabe trabajando en todos los aspectos de forma simultánea.

ID	Nombre	Descripción
RF_12	Monitorización en segundo plano	El sistema debe mantener la monitorización en segundo plano si se ha iniciado independientemente de los análisis del VO2
RF_13	Barra de progreso	El sistema mostrara una barra de progreso para mostrar cuanto tiempo queda para finalizar en análisis activo y a su vez detenerla cuando el usuario entre en reposo.
RF_14	Historial de graficas	El sistema generara graficas con los valores almacenados previamente, tanto frecuencias cardiacas como saturación de oxígeno.
RF_15	Limites superior e inferior de vo2	El sistema debe calcular los valores recomendados máximos y mínimos de saturación de oxígeno basándose en la edad y el género del usuario
RF_16	Historial del mapa	El sistema mostrara un mapa con las mediciones ordenadas geográficamente.
RF_17	Centrado del mapa	El sistema centrara el mapa en la ubicación actual del usuario al abrirlo.

Figura 4.5 Requisitos Funcionales 3º Iteración

ID	Nombre	Descripción
RI_01	Ultima frecuencia en reposo	Frecuencia cardiaca monitorizada en reposo que se usara en el cálculo final del VO2
RI_02	Edad	Edad Introducida por el usuario usada en el cálculo de VO2
RI_03	Peso (kg)	Peso en kg Introducida por el usuario usada en el cálculo de VO2
RI_04	ID Smartwatch	Identificador único del dispositivo smartwatch
RI_05	VO2_Max	Valor final calculado en el análisis de forma activa del usuario.

Figura 4.6 Requisitos No Funcionales 3º Iteración

4.2. Historias de usuario

En esta sección se incluyen las historias de usuario del proyecto, las historias de usuario se modelan a partir de los casos de uso, los casos de uso van a recoger todo el comportamiento del sistema cuando interactúa con el usuario o actor.

4.2.1. Smartwatch

Para la aplicación del Reloj existen los casos de uso que comienzan y terminan los procesos correspondientes a los diferentes análisis, del análisis con actividad física a su vez dependerá el caso de uso del formulario que pide al usuario que introduzca su edad peso y género.

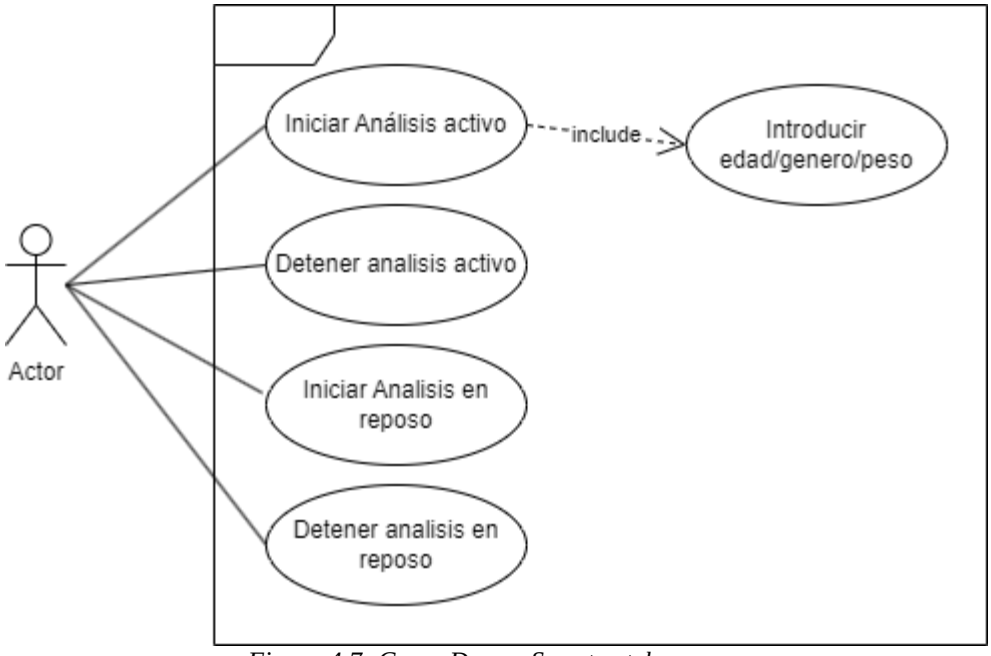


Figura 4.7: Casos De uso Smartwatch

Para estos casos de uso existen las historias de usuario HU_01 – HU_0

ID	HU_01
Nombre	Iniciar Análisis Activo
Prioridad	Alta
Riesgo	Alta
Descripción	Él usuario debe poder iniciar el análisis activo para la medición de oxígeno, el análisis estará bloqueado hasta que el usuario introduzca su edad, peso y género (HU_02) y se haya realizado una medición en reposo anteriormente (HU_04)
Validación	Los hilos de los Sensores comienzan a monitorizar

Figura 4.8: Historia de Usuario HU_01

ID	HU_02
Nombre	Introducir edad/genero/peso
Prioridad	Alta
Riesgo	Medio
Descripción	El usuario introducirá los parámetros que pedirá el sistema para la medición de oxígeno en sangre.
Validación	Se completa el registro y los valores se han almacenado correctamente.

Figura 4.9: Historia de Usuario HU_02

ID	HU_03
Nombre	Detener análisis activo
Prioridad	Alta
Riesgo	Medio
Descripción	El usuario detendrá el análisis activo de Vo2 de forma manual.
Validación	Se detienen los hilos y se reinician los valores de los Listeners

Figura 4.10: Historia de Usuario HU_03

ID	HU_04
Nombre	Iniciar Análisis en reposo
Prioridad	Alta
Riesgo	Alta
Descripción	El usuario comienza el análisis de la frecuencia cardiaca en reposo.
Validación	Se inician los hilos correspondientes al análisis en reposo.

Figura 4.11: Historia de Usuario HU_04

ID	HU_05
Nombre	Detener Análisis en reposo
Prioridad	Alta
Riesgo	Medio
Descripción	El usuario detiene o pausa en análisis en reposo.
Validación	Se paran los hilos y se reinician los valores de los Listeners correspondientes.

Figura 4.12: Historia de Usuario HU_05

4.2.2. Teléfono móvil

Los casos de uso de la aplicación móvil se refieren mayoritariamente a la visualización de los datos recogidos en la aplicación del reloj, en el caso de uso principal HU_09 ver historial de valores, aparte de esto los casos de uso comprenden un tutorial que explica el funcionamiento de la aplicación y el muestreo de datos de forma geográfica en un mapa.

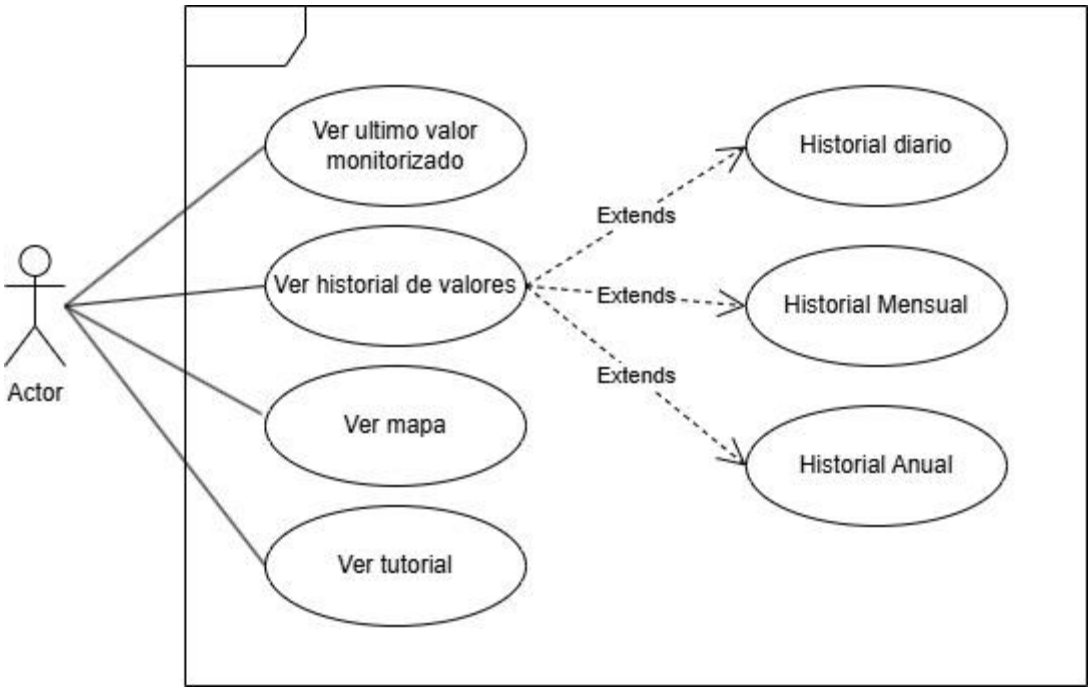


Figura 4.12: Casos de uso aplicación Móvil

ID	HU_06
Nombre	Ver ultimo valor monitorizado
Prioridad	Medio
Riesgo	Medio
Descripción	El usuario detiene o pausa en análisis en reposo.
Validación	Se paran los hilos y se reinician los valores de los Listeners correspondientes.

Figura 4.13: Historia de Usuario HU_06

ID	HU_07
Nombre	Ver mapa
Prioridad	Medio
Riesgo	Alta
Descripción	El usuario visualiza un mapa con su ubicación actual y los valores almacenados geográficamente con su ubicación.
Validación	Se muestran datos correspondientes con los valores en la base de datos

Figura 4.14: Historia de Usuario HU_07

ID	HU_08
Nombre	Ver tutorial
Prioridad	Baja
Riesgo	Baja
Descripción	El usuario accede a la pestaña de tutorial donde se explica el funcionamiento de las aplicaciones.
Validación	La pestaña se visualiza correctamente.

Figura 4.15: Historia de Usuario HU_08

ID	HU_09
Nombre	Ver historial de valores
Prioridad	Alta
Riesgo	Alta
Descripción	El usuario accederá a la pestaña donde se visualizan todo el historial de datos en forma de gráficos.
Validación	Se muestran datos correspondientes con los valores en la base de datos

Figura 4.16: Historia de Usuario HU_09

ID	HU_10
Nombre	Historial diario
Prioridad	Alta
Riesgo	Alta
Descripción	El usuario introduce un día concreto del que se visualizaran los datos con un gráfico de barras
Validación	El grafico se genera correctamente con los datos correspondientes a la base de datos.

Figura 4.17: Historia de usuario HU_10

ID	HU_11
Nombre	Historial mensual
Prioridad	Alta
Riesgo	Alta
Descripción	El usuario introduce un mes concreto del que se visualizaran los datos con un gráfico de barras
Validación	El grafico se genera correctamente con los datos correspondientes a la base de datos.

Figura 4.18: Historia de Usuario HU_11

ID	HU_12
Nombre	Historial Anual
Prioridad	Alta
Riesgo	Alta
Descripción	El usuario introduce un año concreto del que se visualizaran los datos con un gráfico de barras
Validación	El grafico se genera correctamente con los datos correspondientes a la base de datos.

Figura 4.19: Historia de Usuario HU_12

4.3. Diagramas de actividad

En esta sección se presentan los distintos diagramas de actividad generados a lo largo del proyecto. Estos diagramas corresponden a cada una de las historias de usuario identificadas previamente. En términos generales, un diagrama de actividad es un diagrama de flujo que representa las actividades realizadas por el sistema. En este caso, se ilustrará el flujo asociado a cada historia de usuario, acompañado de una breve descripción explicativa de cada diagrama.

HU_01: Iniciar análisis activo

Esta historia corresponde a la funcionalidad principal de la aplicación móvil, el análisis de oxígeno en sangre, que corresponderá al análisis de la actividad física de forma activa, en el diagrama de actividad se representa el flujo que debe seguir el usuario para completar el análisis.

Antes de iniciar el análisis se deben cumplir varias condiciones de las cuales depende el mismo, el usuario debe haber realizado un análisis de la frecuencia cardiaca en reposo previamente (HU_04), además de haber introducido su edad, género y peso (HU_02) si alguno de estos requisitos no se ha cumplido se redirigirá al usuario a los mismos.

Una vez comience el proceso de análisis, se monitorizará la aceleración del reloj para comprobar el estado de actividad constante, durante el periodo de tiempo requerido, al finalizar el periodo se comenzará a leer la frecuencia del usuario que se usara para el cálculo final, si en algún momento se detecta el estado de reposo se pausara y reiniciara el tiempo acumulado además de notificar al usuario.

Una vez finalizado el análisis el valor final se almacenará en el teléfono, si no existe una conexión bluetooth, se almacenará en los servidores de google hasta que se recupere la conexión con el teléfono.

HU_02 Introducir edad/genero/peso

Se pedirá al usuario que introduzca estos valores para el cálculo del oxígeno en sangre, los valores se pedirán siempre que no existan en el sistema, y el usuario tendrá la opción de eliminarlos para introducir nuevos valores.

HU_03 Detener análisis activo

Se asume que el análisis activo ha empezado, el usuario tendrá la opción de pausar y cancelarlo.

HU_04 Iniciar Análisis en reposo

Similar a la historia de usuario HU_01 con la diferencia de que tendrá requisitos previos y se monitorizará la aceleración para comprobar el estado en reposo en vez del estado activo del usuario.

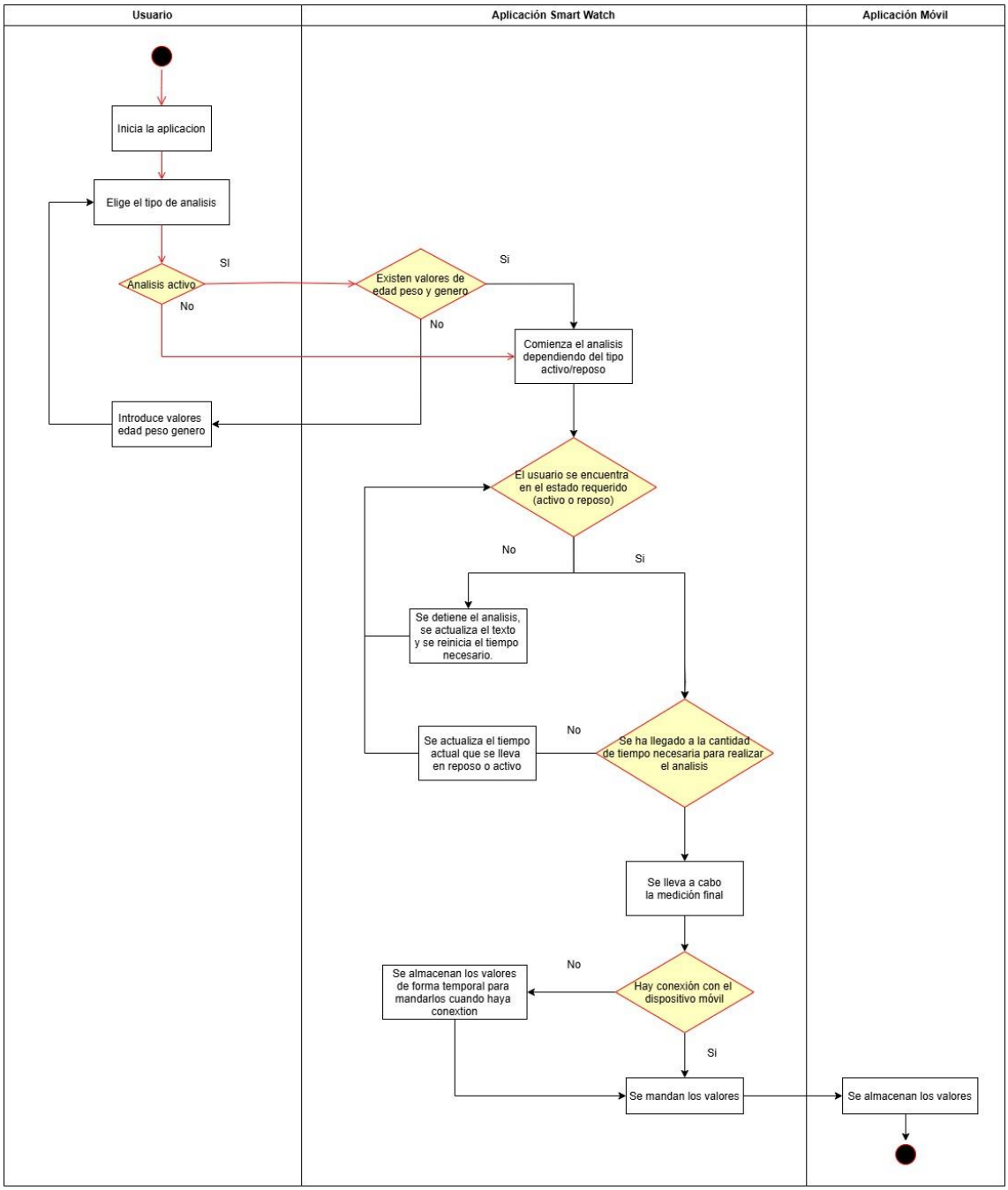


Figura 4.20: Diagrama de actividad Principal

El diagrama de actividad 4.20 incluye las historias de usuario HU_01, HU_02, HU_03 y HU_04

HU_05 Detener Análisis en Reposo

Se asume que el análisis en reposo ha empezado, el usuario tendrá la opción de pausar y cancelarlo.

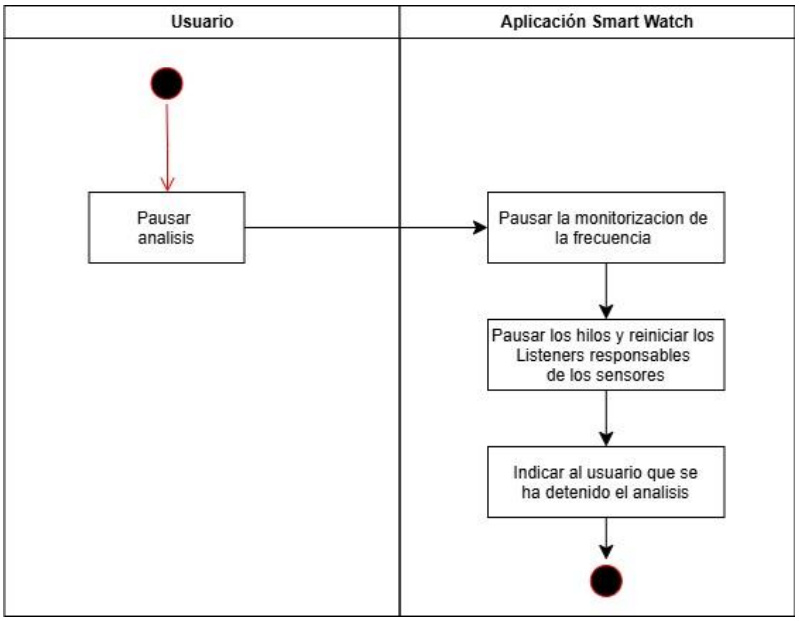


Figura 4.21: Diagrama de actividad Pausa

El diagrama de actividad 4.21 incluye la historia de usuario HU_05

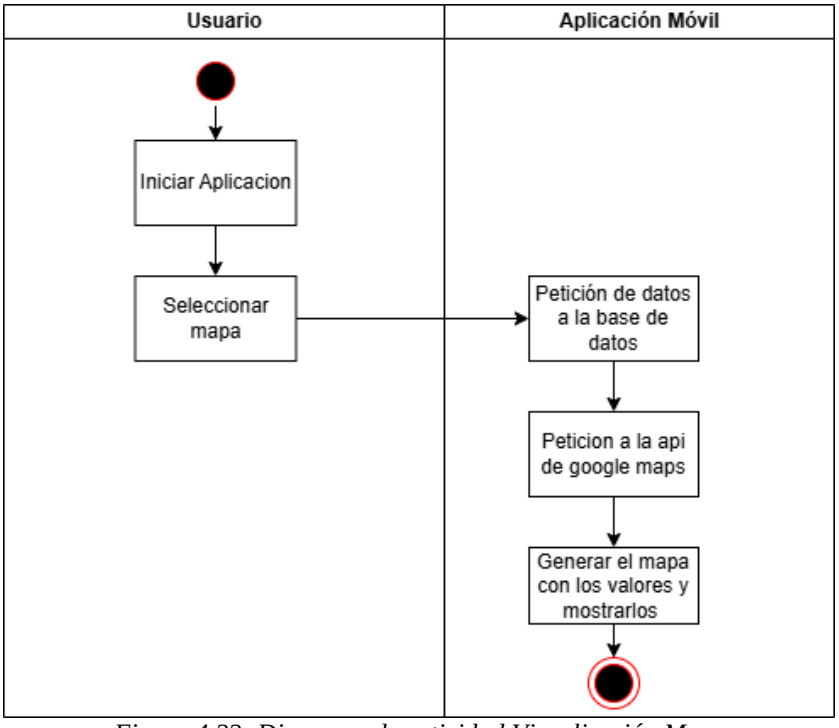


Figura 4.22: Diagrama de actividad Visualización Mapa

El diagrama 4.22 Incluye las historias de usuario HU_07

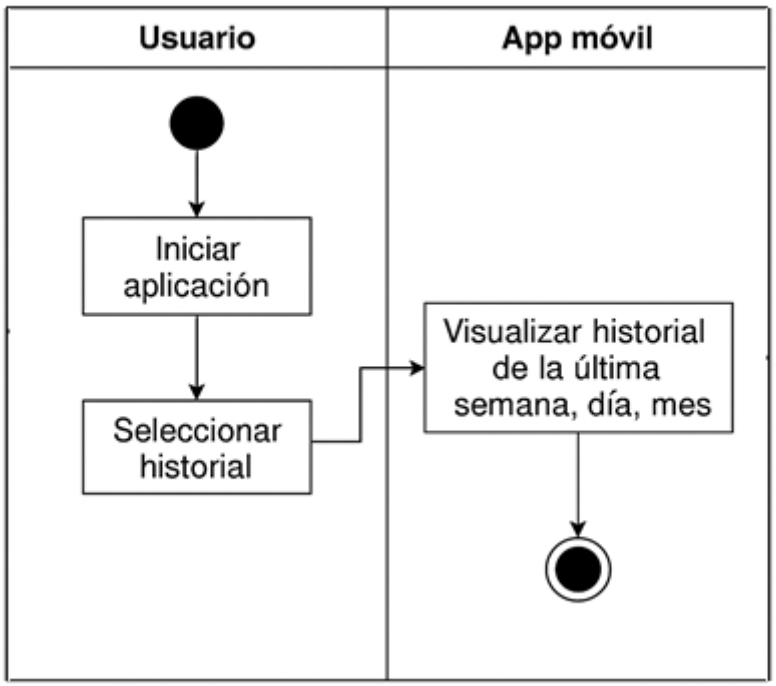


Figura 4.23: Diagrama de actividad Visualización Historial

El diagrama 4.23 Incluye las historias de usuario HU_06-HU_12

HU_06 Ver Ultimo valor monitorizado

Al iniciar la aplicación móvil se visualizará el ultimo valor monitorizado, ya sea un valor de frecuencia cardiaca o saturación de oxígeno, se mostrará su valor, su ubicación y la fecha del análisis.

HU_09 - HU_12: Ver Historial de valores

La pestaña del historial engloba las historias de usuario HU-09 hasta HU-12, las funcionalidades de visualizar los datos almacenados en la aplicación, mediante gráficos, donde varían los intervalos de fechas que se quieren visualizar, días, meses, años.

HU_07 Ver Mapa

Los valores se almacenarán con la ubicación donde fueron medidos, se mostrarán de forma geográfica con un mapa mediante el api de Google Maps donde cada pop up representara un valor almacenado.

HU_08 Ver Tutorial

Se mostrará un tutorial al seleccionar la pestaña en la aplicación móvil.

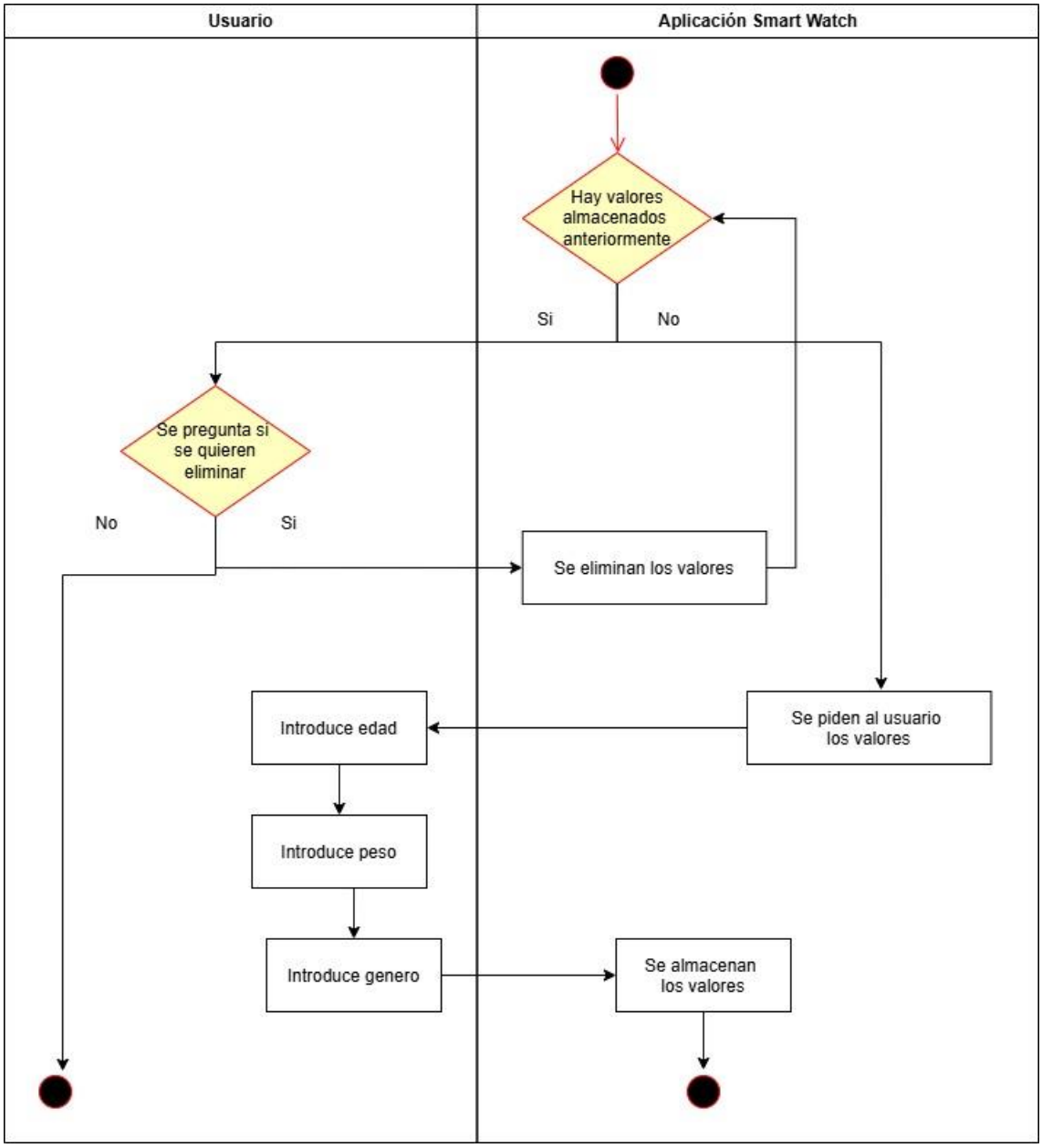


Figura 4.22: Diagrama de actividad Guardado de Datos

El diagrama de actividad 4.22 Muestra en detalle la actividad de la historia de usuario HU_02

4.4. Modelo de dominio

A continuación, se muestra el modelo de dominio que representa conceptualmente todos los elementos involucrados en el sistema, éste gráfico, *figura 4.20*, describe las distintas entidades que forman parte del sistema, junto con sus características, roles y como se relacionan entre sí con respecto al comportamiento del sistema.

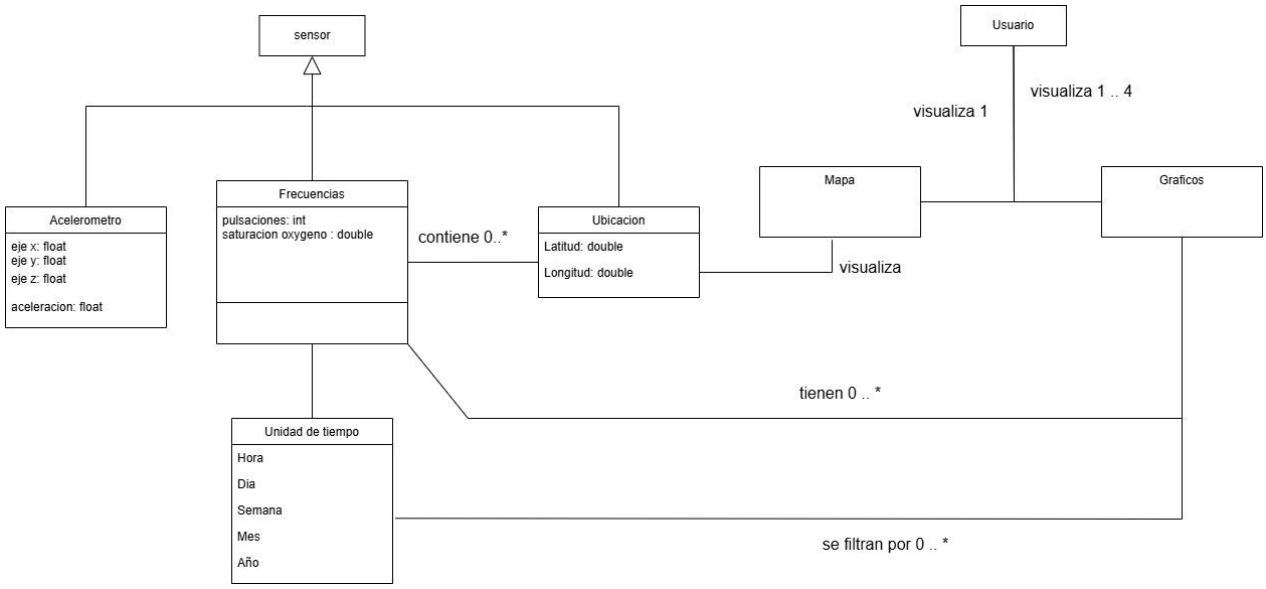


Figura 4.23 Modelo del dominio

- La clase **Usuario** se relaciona directamente con las visualizaciones del sistema. Cada usuario puede visualizar una única instancia del **Mapa**, en el cual se representan las ubicaciones registradas, y entre uno y cuatro **Gráficos**, que muestran las frecuencias, agrupadas en función del tiempo, por ejemplo, los datos de un día, un mes o un año.
- El **Mapa** permite la visualización de las instancias de la clase **Ubicación**, lo que facilita representar gráficamente el lugar donde se han tomado las mediciones correspondientes.
- La clase **Frecuencias**, que agrupa datos como las pulsaciones y la saturación de oxígeno, puede estar asociada con múltiples ubicaciones, lo que permite rastrear los valores fisiológicos en función del lugar donde fueron capturados.
- Además, las frecuencias pueden ser **filtradas por diferentes unidades de tiempo**, tales como hora, día, semana, mes o año. Esta relación permite al usuario analizar la evolución de los datos en distintos marcos temporales.
- Finalmente, las clases **Acelerómetro**, **Frecuencias** y **Ubicación** comparten una relación de herencia desde una clase abstracta denominada **Sensor**, lo que indica que todas estas entidades representan distintos tipos de sensores utilizados para la recogida de datos en el sistema.

Capítulo 5

Diseño

En este capítulo se cubrirá el diseño del proyecto, la arquitectura y los patrones de las aplicaciones, el diseño de datos y la estructura general del código junto con los diagramas que representan secciones específicas del desarrollo, como el diseño de datos o los diagramas de despliegue.

5.1. Diseño de datos

La estructura de datos de la base de datos se compone de una tabla simple detallada en la figura 5.1, inicialmente se consideraron dos tablas una para los datos relacionados con las mediciones de frecuencia cardiacas y otra para los análisis de oxígeno en sangre.

Finalmente se optó por una única tabla ya que ambos van a compartir la mayoría de sus atributos y se simplifica mucho la inserción de datos a la base de datos, se incluye también un atributo “ActivityType” que realizara la distinción del tipo de actividad a la hora de tratar con datos en el sistema.

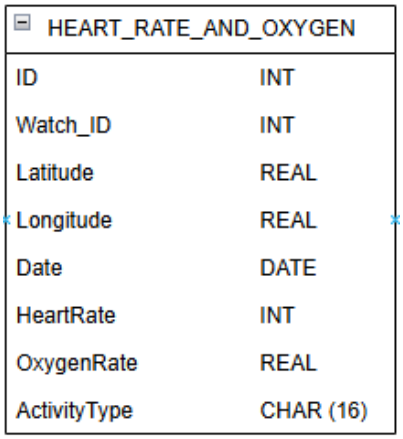


Figura 5.2: Diagrama Entidades Base de datos

5.2. Arquitectura

5.2.1. App reloj

Se sigue las recomendaciones de diseño de la documentación de Android developers

la arquitectura se basa en un diseño de 3 capas, de usuario o capa de interfaz (UI), capa de dominio y capa de datos.

Al separar las capas se facilita la escalabilidad de los componentes, la reutilización y flexibilidad del código y se obtiene una organización más limpia de cara a la hora de separar y compartimentalizar elementos de acuerdo a sus funcionalidades.

- **Capa de usuario o interfaz:** La capa responsable de la interacción del usuario con la aplicación, en Wear OS la compondrán fragmentos y actividades, la actividad principal *MainActivity* que contendrá instancias de cada uno de los fragmentos que representan las funcionalidades de la aplicación.

- **Capa de dominio:** Capa que gestiona la comunicación entre los fragmentos y contendrá las funcionalidades del proyecto, en el caso de este proyecto existirán hilos y ciclos de vida concurrentes, por lo que conviene separarlos para evitar complicaciones, la conformaran los hilos correspondientes a los sensores, acelerómetros y responsables de la ubicación.
- **Capa de datos:** La capa de datos contiene a las clases que forman el modelo de datos y las clases responsables de enviar los datos a la base de datos en la aplicación móvil, existirán modelos de esta base de datos contenidos en el paquete común entre las dos apps, *Utilities.*

5.2.2. App móvil

La arquitectura de la aplicación móvil es la misma que la del reloj con la diferencia de que no cuenta con una capa de dominio, ya que se trata de una capa opcional que suele ser recomendable en la mayoría de los casos.

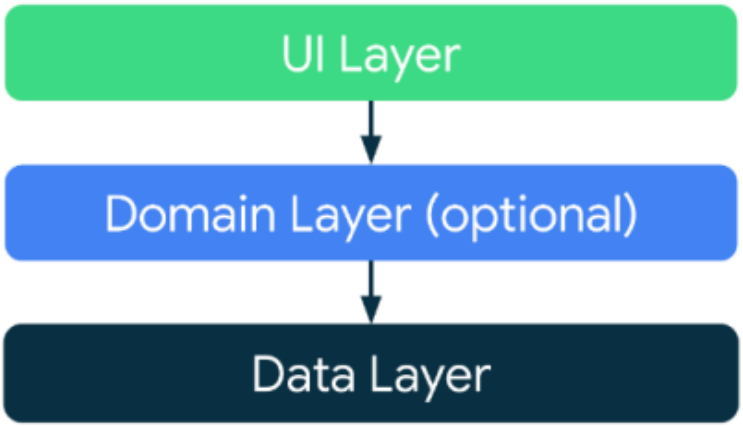


Figura 5.2: Arquitectura en 3 capas de una aplicación Android [44]

Debido a la simplicidad de la aplicación móvil y que la gran mayoría del desarrollo se centra en interfaces en forma de gráficos y el tratamiento de datos en la base de datos en forma de consultas a la misma no se considera necesaria la capa de dominio.

- **Capa de usuario o interfaz:** Contiene la navegación entre las diferentes pestañas de la aplicación móvil por medio de una barra de navegación, los fragmentos que contienen las diferentes gráficas, el fragmento del mapa y el fragmento del tutorial.
- **Capa de datos:** La forman todas las clases correspondientes a la base de datos y su funcionamiento, todas las consultas y la creación de la base de datos estarán contenidas en una misma clase *db_TFG* y los modelos de datos que se encuentran el paquete *Utilities*.

5.3. Patrones de diseño

Los patrones de diseño son técnicas de desarrollo enfocadas al desarrollo de problemas comunes, esta sección engloba los diferentes patrones usados durante el desarrollo del proyecto. [20]

5.3.1. Singleton

El patrón singleton o patrón de instancia única, garantiza que solo exista una instancia única para una clase y la creación de un mecanismo de acceso global a esta.

La clase en el proyecto es *SharedData*, que será común para las dos aplicaciones móvil y reloj, se encuentra en el paquete *Utilities*, en esta clase cumplirá funciones auxiliares durante todos los procesos análisis, con atributos para controlar el funcionamiento de los hilos, además de simplificar algunas consultas a la base de datos, en la app móvil a la hora de mostrar datos.

5.3.2. Observer

El patrón *Observer* es un patrón de diseño de comportamiento que permite que un objeto observable, gestione una lista de objetos *observable* que van a depender del *observer*, cuando el *observable* experimenta algún cambio se notificara a los *observers*.

En el caso de la aplicación del reloj, los sensores, como los acelerómetros y el monitor de frecuencia cardíaca, emplean este patrón de manera implícita. Aquí, el objeto *SensorManager* actúa como el observable, almacenando una lista de *Observables*, los *SensorListeners* u observers, que recibirán las notificaciones.

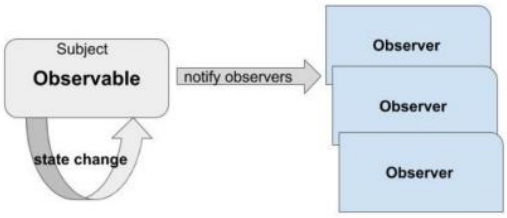


Figura 5.3: Patrón Observer [46]

Cuando un sensor del dispositivo detecta un cambio en su estado, el *SensorManager* alerta automáticamente a todos los *SensorListeners* registrados para ese sensor, llamando a su método `onSensorChanged`. A través de este método, los *SensorListeners* pueden procesar los nuevos datos del sensor y realizar las actualizaciones necesarias en la aplicación.

De esta forma, el patrón Observer facilita que la aplicación Android reciba actualizaciones de los sensores de manera asíncrona, eliminando la necesidad de verificar constantemente su estado en un bucle continuo. En su lugar, el *SensorManager* se encarga de informar a los oyentes cuando ocurre un cambio en el sensor. Una descripción más detallada de la API de sensores será abordada en un capítulo posterior.

5.4. Diagramas de paquetes.

Los diagramas de paquetes muestran las relaciones de dependencia entre los distintos paquetes que formal el proyecto, Habrá un diagrama de paquetes para cada una de las dos aplicaciones.

Al igual que en la arquitectura, los diagramas de paquetes se van a agrupar por capas, 3 en la aplicación del reloj y 2 en la del móvil.

Figura 5.2: Arquitectura en 3 capas de una aplicación Android

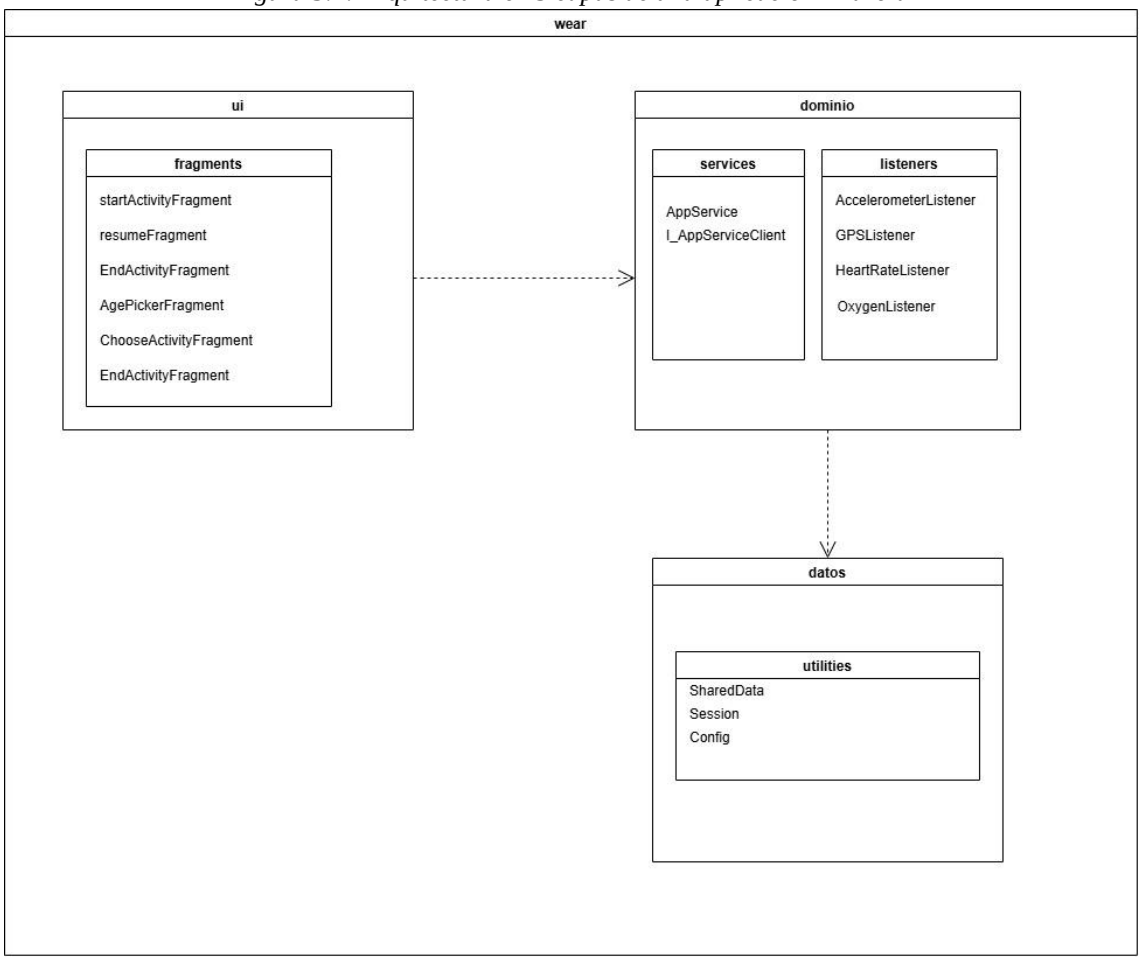
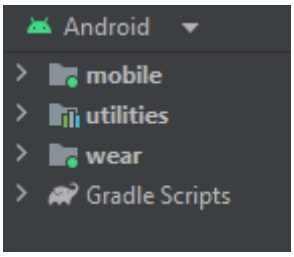


Figura 5.4: Diagrama de Paquetes

En la aplicación wear la interfaz de usuario es en forma de *fragments* mientras que la lógica en el paquete de dominio está conformada por los Listeners de los sensores y los servicios con *AppService* y su interfaz que gestionaran los Listeners y la comunicación entre ellos.

5.5. Estructura del proyecto

En esta sección se comenta como se han distribuidos los componentes de las aplicaciones.



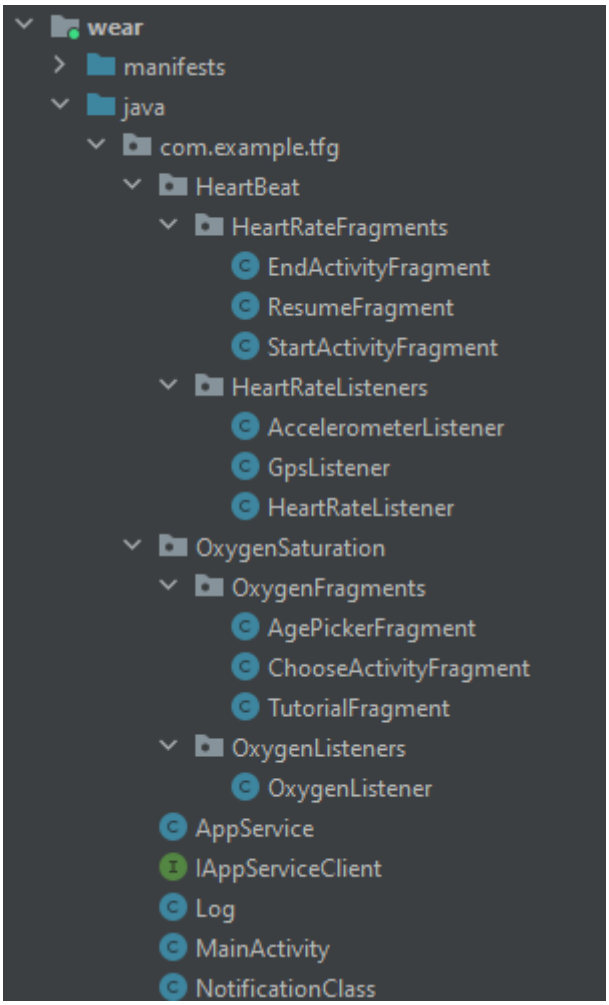
Principalmente el codigo Java de android de cada proyecto se divide en dos secciones.

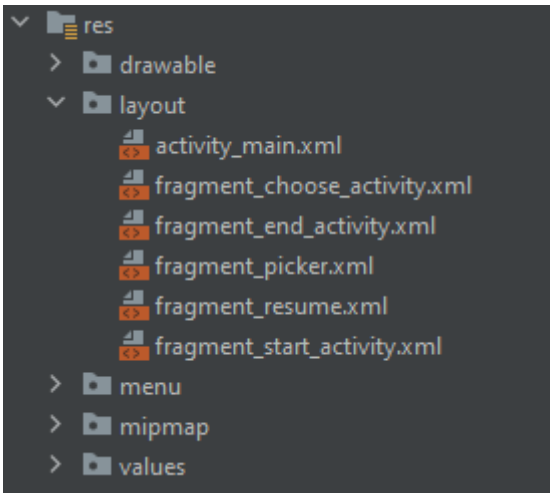
El codigo java, cuyas clases se encuentran en la carpeta del mismo nombe, serán la funcionalidad del proyecto.

Y la carpeta res que contiene todo lo relativo a la interfaz, dentro de esta,

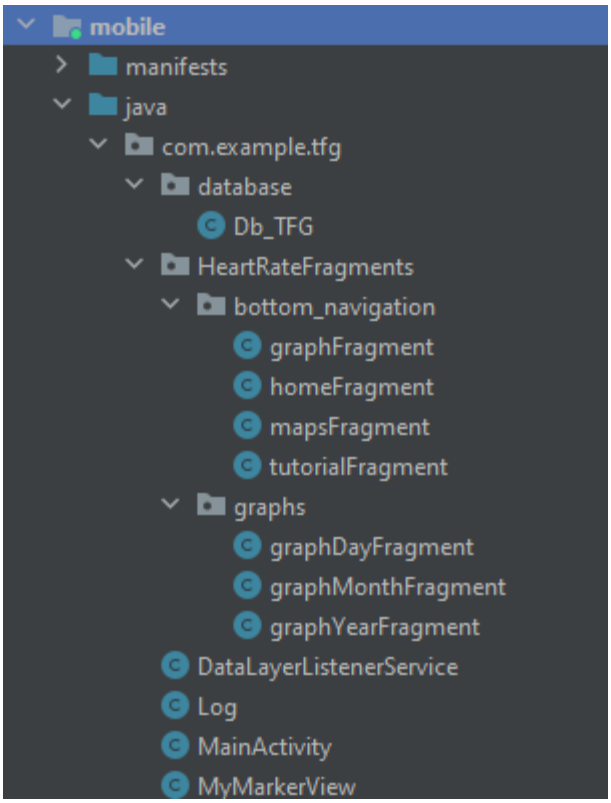
- La carpeta **layout** contendra los ficheros XML que corresponden a los fragmentos y la principal forma de diseñar la interfaz de las aplicaciones.
- Drawable: Imágenes y archivos que se usan en el resto de las clases.
- Minimap: Carpetas que contienen la miniatura de las imágenes a lo largo del proyecto en diferente resolucion, se usa por ejemplo en el icono de la aplicación movil.
- Values: Valores y constantes que se usan a lo largo de las aplicaciones, por ejemplo, Strings que representan el valor rbg de colores, listas de Strings que permiten al usuario elegir varias de la misma lista en un scroller, en general valores que facilitan la reutilizacion de codigo y evitan el uso de cadenas de texto en el codigo.

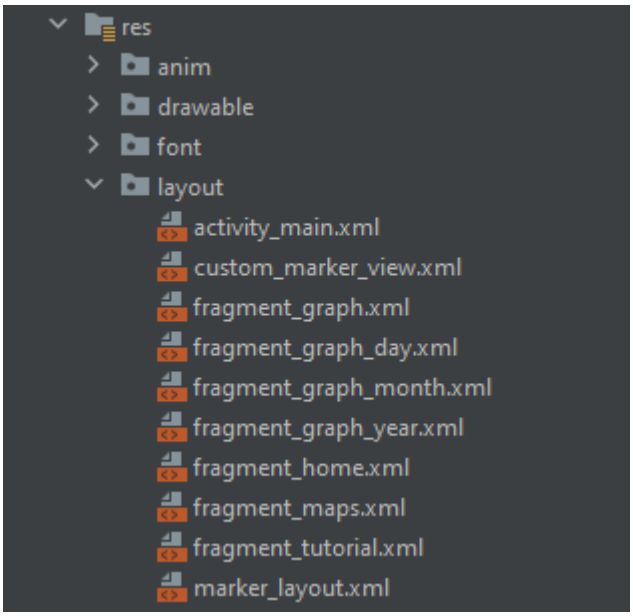
5.5.1. App del reloj



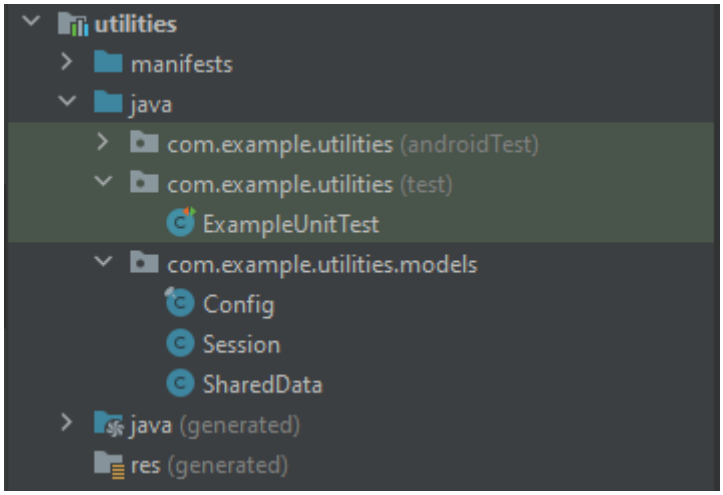


5.5.2. App móvil





5.5.3. Paquete común: Utilities



5.6. Diagrama de despliegue

El diagrama de despliegue se utiliza para ilustrar cómo se distribuyen físicamente los componentes del software a través de los distintos nodos de la red. En este caso, se identifican cuatro componentes que abarcan ambas aplicaciones:

-Los dos dispositivos móviles (el teléfono y el reloj)

-Los servidores de Google, que facilitan la sincronización de los datos enviados del reloj al teléfono.

-La base de datos, que es local, está almacenada en el teléfono.

No se ha realizado una distinción de los diagramas para cada aplicación debido a la simplicidad de los mismos. En términos generales, los dos componentes situados a la izquierda corresponderían al diagrama de despliegue de la aplicación Wear OS, mientras el resto, excepto el reloj, formarían parte del diagrama de despliegue de la aplicación Android del teléfono.

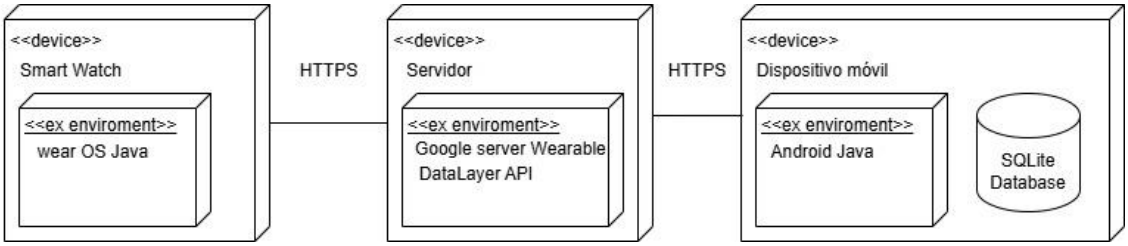


Figura 5.11: Diagrama de Despliegue

Capítulo 6

Implementación

6.1 Posibilidades para la monitorización del oxígeno en sangre

A continuación, se analizarán las opciones disponibles para el análisis de la saturación de oxígeno, el planteamiento inicial, las formulas disponibles y la opción por la que se optó finalmente.

Planteamiento inicial

Inicialmente se pretendía monitorizar la saturación de oxígeno en sangre directamente con el sensor del reloj que ofrece el SDK de Samsung health sensor [40], el problema es que el acceso a los datos y el uso de este sensor es privado y está limitado al programa de partners de Samsung.

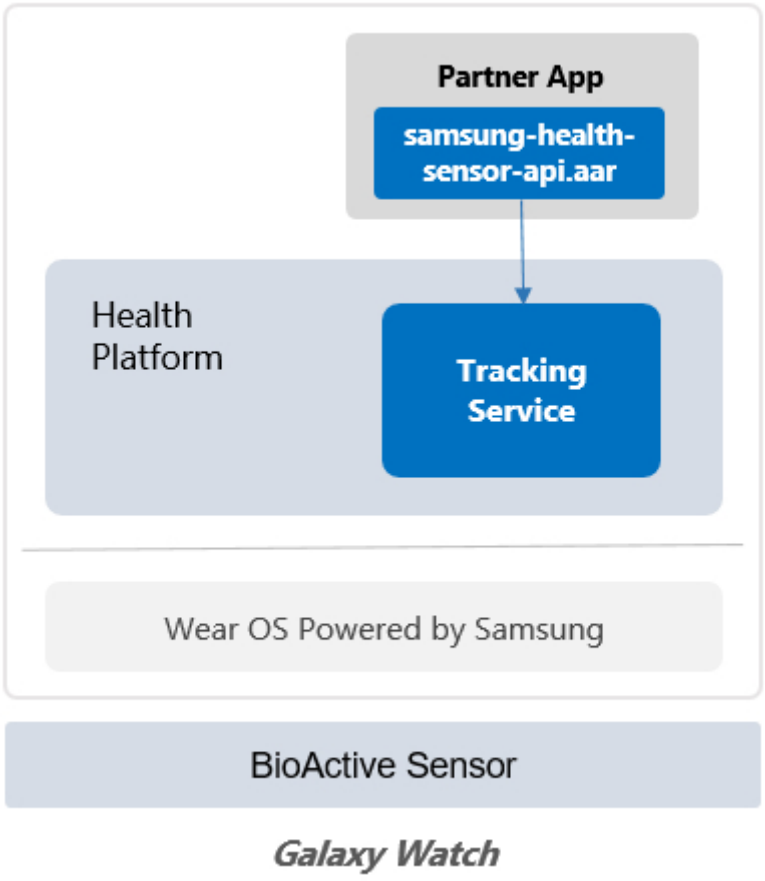


Figura 6.1 Arquitectura de la API de Samsung [45]

Alternativas que se plantearon.

Acceder a los datos del sensor sin procesar con las APIs REST de google fit [41]: Teóricamente se podrían realizar lecturas de saturación de oxigeno usando la API sensor de google fit, esta alternativa se descartó ya que los inconvenientes que implica son demasiado severos.

La API dejara de estar disponible a partir del 30 de junio de 2025, no solo eso, sino que los datos se obtienen sin procesar lo que conlleva una gran dificultad a la hora de trabajar con ellos.

Plantear el consumo de oxigeno como mililitros por kilogramos por minuto (ml/kg/min):

Ya que no se dispone de un sensor para la medición directa, se considera el volumen de oxigeno que el cuerpo utiliza cada minuto en plena actividad física, o VO2, para el cálculo de este valor existen múltiples fórmulas que hacen uso de la frecuencia cardiaca durante el periodo de actividad y la frecuencia cardiaca del usuario en reposo, estos valores si se pueden medir con los sensores de los que disponemos usando la API de Android.

Métodos para el cálculo del VO2.

El método más preciso sería usando una máscara de oxígeno que lee con total precisión el consumo del usuario, obviamente no se dispone de esta opción y perdería el sentido de ser una app de Smart watch, pero es interesante mencionarla.

Existen varias pruebas para calcular el Vo2 que requerirán actividad física por parte del usuario, teniendo en cuenta como norma general que cuanto más al límite se lleva el cuerpo más precisa será la medición, así mismo, cada test requerirá de un tiempo mínimo para ser fiable, este ha sido el principal factor a considerar a la hora de elegir la fórmula para el cálculo, ya que idealmente se le pedirá el mínimo tiempo al usuario para este test.

Test Simplificado

- VO2 max = 15,3 x (FCM/RHR)

Donde:

- FCM = Frecuencia cardiaca máxima en 20 segundos multiplicada por 3.
- RHR = Frecuencia cardiaca en reposo en 20 segundos multiplicada por 3.

Otra versión de la formula simplificada que tiene en cuenta la edad puede ser:

- VO2 max = 15.3 x MHR / RHR

Donde MHR = 208 – (0.7 x Edad)

Estas fórmulas simples que no tiene en cuenta factores como el peso o el género y exigen un tiempo mínimo de actividad por lo que contaran con una precisión menos fiable por lo cual se descartan.

Test de Cooper

Una de las pruebas más sencillas y fiables, consiste en que el usuario recorra la distancia máxima posible en un periodo de 12 minutos.

- VO2 max= 0,0268 x Distancia (m) -11,3

En el caso de nuestra aplicación la simplicidad no es un factor que realmente influya ya que el cálculo lo realizara el sistema y no implica el usuario, mientras que un tiempo de 12 minutos por cada prueba resulta inconveniente para el usuario, teniendo en cuenta la precisión de las siguientes pruebas.

Test de Course Navette

Esta prueba consiste en seguir el ritmo de una grabación con un audio que marca el ritmo de la carrera que hay que seguir, el ritmo incrementara progresivamente y el usuario deberá continuar hasta que no consiga completar la distancia de 20 metros a tiempo, el vo2 se determina con el valor de la última velocidad alcanzada.

. VO2max = 5,857 x Velocidad (Km/h) – 19,458

Este test está más orientado a deportistas y aunque cuenta con una gran fiabilidad la duración puede extenderse hasta 23 minutos, por lo que no se tendrá en cuenta para el proyecto.

Test de Rockport

El test consiste en completar una distancia de 1.600 metros andando y la formula tendrá en cuenta el peso corporal, la edad, el género y el tiempo.

-VO2max = 132,6 – (0,17 x Peso Corporal) – (0,39 x Edad) + (6,31 x Genero)

– (3,27 x Tiempo) – (0,156 x Frecuencia cardiaca media)

El método que se usara en el proyecto es una modificación al test de Rockport donde se fijara el tiempo mínimo que el usuario permanecerá de forma activa, no se tendrá en cuenta la distancia mínima y se recomendara actividad de alta intensidad en vez de recorrer una distancia ya que se busca la mayor frecuencia cardiaca

El valor obtenido se interpreta con los valores de esta tabla en función a la edad y el sexo del usuario.

Sexo	Edad	Pobre	Justo	Medio	Bueno	Excelente
Hombre	Por debajo de 29	< 24.9	25-33.9	34-43.9	44-52.9	Superior a 53
	30-39	< 22.9	23-30.9	31-41.9	42.49.9	Superior a 50
	40-49	< 19.9	20-26.9	27-38.9	39.44.9	Superior a 45
	50-59	< 17.9	18-24.9	25-37.9	38.42.9	Superior a 43
	60-69	< 15.9	16-22.9	23-35.9	36-40.9	Superior a 41
Mujer	Por debajo de 29	< 23.9	24-30.9	31-38.9	39-48.9	Superior a 49
	30-39	< 19.9	20-27.9	28-36.9	37.44.9	Superior a 45
	40-49	< 16.9	17-24.9	25-34.9	35-41.9	Superior a 42
	50-59	< 14.9	15-21.9	22-33.9	34-39.9	Superior a 40
	60-69	< 12.9	13-20.9	21-32.9	33-36.9	Superior a 37

Figura 6.2: Tabla de Valores de la Saturación de Oxigeno

6.1.1. Sensores Frecuencia cardiaca y Acelerómetro

La monitorización de la frecuencia cardiaca se llevará a cabo con los sensores de la API de Android, que tendrán las clases propias *Sensor* y *SensorManager* e implementan la interfaz *SensorEventListener*, esta interfaz cumple dos funciones, comprobar cuando se realizan cambios en el sensor y la precisión del mismo, con los métodos *onSensorChanged* y *onAccuracyChanged*.

En el *SensorManger* se inicializarán los listeners que se encargaran de llamar a los métodos que comprueban si se han realizado los cambios en el sensor, con el método *registerListener*, de igual forma que cuando se finaliza el flujo de la funcionalidad del listener se eliminaran con *unregisterListener*.

Cada sensor cuenta con los siguientes parámetros que se determinan en el momento que se registra el Listener.

- SensorEventListener: el listener que recibirá los datos monitorizados por el sensor.
- Sensor: el sensor que se va a monitorizar. En nuestro caso este será el sensor del acelerómetro o el de la frecuencia cardiaca.
- Tasa de refresco: Es un valor entero que define cada cuanto tiempo se va a llamar al método onSensorChanged del listener, este valor se expresa en microsegundos ($1\ \mu s = 10^{-6}$ s). Existen varios valores predeterminados que están detallados en la tabla. En dicha tabla se indican cada uno de estos valores junto con su refresco aproximado en microsegundos.

Valor	Delay (en microsegundos)
SENSOR_DELAY_NORMAL	200000
SENSOR_DELAY_UI	60000
SENSOR_DELAY_GAME	20000
SENSOR_DELAY_FASTEST	0

Figura 6.2: Tabla de Valores Predeterminados del Delay en el muestreo de los sensores

Para todos los sensores de la aplicación se usará una tasa de refresco SENSOR_DELAY_NORMAL excepto para el listener de la frecuencia cardiaca que se usa SENSOR_DELAY_FASTEST.

6.1.2. Ubicación

La detección de la ubicación se realizará desde el reloj, se plantearon dos posibilidades para la monitorización de la ubicación.

API FusedLocationProvider: La API más popular de google para la gestión de ubicaciones, hace uso de sensores de localización inteligentes.[22]

API LocationManager: Una clase perteneciente a la Api de Android, más enfocada a el desarrollo Android y más simplificada para el uso de una aplicación Wear OS[23]

Ambas opciones cumplen la misma función y se podrían utilizar, se va a optar por la segunda al ser más fácil de integrar y encontrar más documentación y ejemplos praticos.

La clase **LocationManager**, que proporciona los métodos y variables necesarias para monitorizar la ubicación. Similar al caso de los sensores, es necesario utilizar un listener encargado de recibir las actualizaciones de ubicación. El método principal de este listener es **onLocationChanged**, que se ejecuta cada vez que se recibe una nueva ubicación.

En el monitoreo de la ubicación, algunos parámetros clave están relacionados con la precisión y el intervalo entre mediciones consecutivas. El método **requestLocationUpdates** de la clase **LocationManager** gestiona estas actualizaciones. Este método recibe varios parámetros, que se describen a continuación:

- **Provider**: Define las señales utilizadas para determinar la ubicación. Este parámetro es crucial, ya que de la elección del provider van a depender las condiciones del entorno, factores como, por ejemplo, la red y los satélites que se usara para obtener la ubicación.
- **Tiempo mínimo**: Especifica el intervalo mínimo entre actualizaciones de ubicación, medido en milisegundos.
- **Distancia mínima**: Establece la distancia mínima requerida entre actualizaciones, medida en metros.
- **Location listener**: Es el listener que gestionará las actualizaciones de ubicación. Este componente incluye gran parte de la lógica relacionada con la ubicación dentro de la aplicación.

La elección del provider adecuado depende de varios factores, como la disponibilidad de satélites o de conexión a internet. En interiores, los satélites suelen tener un rendimiento deficiente, por lo que el **network provider** (basado en redes Wifi o móviles) suele ser más efectivo. Por el contrario, en exteriores, el **gps provider** es generalmente la mejor opción, ya que las conexiones Wifi o de datos móviles pueden no estar disponibles.

Cabe destacar que muchos relojes, como el utilizado en este proyecto, no cuentan con una tarjeta SIM, lo que limita el uso del **network provider** a las redes Wifi.

En este proyecto se usará el método **getBestProvider** de **LocationManager** que determina la configuración optima del provider de forma automática.

6.2. Detección de la actividad

La detección de la actividad verifica que el usuario se encuentra en el estado previo a realizar cualquier análisis, en la versión anterior del proyecto solo se comprobaba el estado en reposo del usuario para la medición de la frecuencia cardiaca de forma periódica cada 30 minutos en segundo plano.

En esta versión se verificarán los estados de reposo para un análisis puntual o la actividad física constante durante un periodo de tiempo para un análisis activo.

6.2.1. Flujo de detección de actividad

La detección del estado activo del usuario se realiza mediante los acelerómetros del reloj, inicialmente se planteó comprobar los cambios en el acelerómetro durante la duración del análisis, con un margen 1-2 segundos en caso de que el usuario pudiese parar momentáneamente sin cancelar el análisis, este planteamiento no sería erróneo, pero presenta un problema.

El valor de saturación de oxigeno va a ser más preciso cuanto más intensa sea la actividad física, dentro del tiempo delimitado, se llegó a la conclusión mediante pruebas que la forma óptima de realizar el análisis es comprobar la actividad constante del usuario durante el tiempo mínimo delimitado, 40 segundos sin realizar ninguna lectura y una vez se ha completado el tiempo se empieza a medir la frecuencia del usuario durante un breve periodo para el cálculo del oxígeno.

1- Si en el reloj no están almacenadas la edad, peso y genero del usuario se le pedirá al usuario antes de que comience el análisis, si no, empieza el proceso.

2- Se inicia una barra de progreso que muestra al usuario el tiempo restante en actividad, solo se considerara estado activo si el valor del acelerómetro está constantemente cambiando, si el usuario entra en reposo en cualquier momento se le notificara en la pantalla y la barra de progreso parara, si el estado de reposo persiste el tiempo y el análisis se reiniciaran.

3- Una vez se haya completado el periodo en actividad, se deja de monitorizar los acelerómetros y se inician los sensores de monitorización, en este punto la actividad del usuario no es necesaria, pero si recomendable, entrar en reposo no parara el análisis, pero tampoco se le dice al usuario que tenga que parar, una vez termina el periodo de análisis se avisa al usuario de que el análisis a finalizado.

4- Se finaliza el análisis, transmiten los datos a la base de datos y detienen los hilos.

6.2.2. Comunicación y gestión entre hilos para la detección de actividad

La detección del estado del usuario, activa o en reposo, detectada mediante el sensor del acelerómetro del reloj se va a gestionar con un hilo o thread, la detección en reposo va a continuar periódicamente cada 30 minutos en segundo plano mientras que la detección activa va a ser un proceso único, por esto se utiliza un hilo para cada una, de forma similar va a haber un hilo para el sensor de frecuencia cardiaca y uno para la ubicación.

La coordinación y comunicación entre hilos es esencial para el funcionamiento correcto de la aplicación de smartwatch, hay que asegurarse que no ocurre el solapamiento de lecturas en el sensor de la frecuencia cardiaca o del acelerómetro.

Esta comunicación se consigue implementando un *bound service* [23]

Para cada hilo, estos servicios implementan una interfaz *cliente-servidor*, esto significa que cada funcionalidad, en nuestro caso actividades, tendrá vinculada un servicio, las actividades van a actuar como el cliente de esta relación, y el servicio como servidor, lo que permite que el *bound Service* se encargue de enviar y responder peticiones entre otros servicios para realizar la comunicación entre funcionalidades y sus servicios.

La segunda parte de la comunicación van a ser las clases de tipo *Handler*, esta clase de [24] cumple dos funciones, programar mensajes e hilos o *runnables* para que sean mandados a otros hilos, ya sea con un delay, o de forma recursiva, asi como la comunicación directa con otros hilos.

Por ejemplo, se podría mandar un mensaje desde el hilo del acelerómetro al hilo que gestiona el sensor de la frecuencia cardiaca para que comience un análisis una vez se ha comprobado el estado de reposo o al contrario mandar un mensaje para que se detenga la lectura de la frecuencia si se detecta un cambio en el estado.

6.3. Gestión de datos.

Esta sección detalla la gestión de datos, concretamente la transferencia de los datos medidos desde el reloj al teléfono y su almacenamiento.

6.3.1. Transmisión de los datos

Una vez a concluido cualquiera de los procesos de reloj sin incidencias, los datos se transfieren al teléfono móvil vinculado, esta transferencia se realiza mediante la API **Wearable Data Layer** [25] esta api parte de los servicios de google, está diseñada para la comunicación entre dispositivos Wear OS y dispositivos Android y no es compatible con otros dispositivos wearables que no tengan Wear OS.

Existen dos opciones para la comunicación, detalladas en la documentación oficial, de forma directa entre el reloj y el móvil mediante Bluetooth o mediante los servidores de google a través de una red a la que los dos dispositivos estén conectados, normalmente wifi o LTE.

Dentro de la API existen varios objetos con diferentes casos de uso detallados en la documentación.

Message Client: Su función principal es enviar mensajes a nodos que estén conectados en el momento. Sin embargo, los objetos de este tipo tienen una limitación importante, los nodos deben estar conectados para que los mensajes se envíen y reciban correctamente. Además, no permite enviar datos que superen los 100 kb de tamaño.

Channel Client: Disponible únicamente a partir de Wear OS 2.0, este objeto permite enviar datos mayores a 100 KB y de diversos tipos, no solo mensajes. Su principal ventaja frente a Message Client es esta capacidad de manejar datos más grandes. Sin embargo, está diseñada para comunicación en tiempo real, lo que significa que ambas aplicaciones deben estar activas para que la comunicación sea efectiva.

Data Client: Este es el objeto elegido para este proyecto. Su principal ventaja es que permite el envío de datos de manera asíncrona, es decir, no es necesario que ambos nodos estén conectados simultáneamente. Al igual que Channel Client, admite el envío de datos de gran tamaño. En casos donde los datos son muy grandes, es recomendable usar un objeto de la clase *Asset* y asociarlo a un objeto de la clase *DataItem*. Sin embargo, este no es el caso de los datos del proyecto.

La transferencia de datos de forma práctica se resume de la siguiente forma:

Se crea una instancia de la clase *PutDataMapRequest* [26]

al que se le pasa la dirección especifica que corresponde al canal de comunicación del dispositivo móvil, después se añaden a la instancia los datos que se quieren transmitir dependiendo de la actividad realizada, se crea un objeto de la clase *PutDataRquest* que utiliza la instancia anterior para formatearla como una petición para la transferencia de

datos, esta se manda de forma automática a los nodos conectados, después se comprueba si la transferencia se ha realizado de forma exitosa mediante dos Listeners.

```
private void sendData(double latitude, double longitude, String date, ArrayList<Integer> heartRatesList, String activityType, double vo2Value
                      float[] oxygenLimits) {
    PutDataMapRequest dataMap = PutDataMapRequest.create("/TFG_path");
    dataMap.getDataMap().putString("ID_Watch", Settings.Secure.getString(
            getApplicationContext().getContentResolver(),
            Settings.Secure.ANDROID_ID));
    dataMap.getDataMap().putString("date", date);
    dataMap.getDataMap().putDouble("latitude", latitude);
    dataMap.getDataMap().putDouble("longitude", longitude);
    dataMap.getDataMap().putIntegerArrayList("heartRatesList", heartRatesList);

    dataMap.getDataMap().putString("activityType", activityType);
    dataMap.getDataMap().putDouble("vo2Value", vo2Value);
    dataMap.getDataMap().putFloat("lowerLimit", oxygenLimits[0]);
    dataMap.getDataMap().putFloat("upperLimit", oxygenLimits[1]);

    PutDataRequest request = dataMap.asPutDataRequest();
    request.setUrgent();

    Task<DataItem> dataItemTask = Wearable.getDataClient( activity: this).putDataItem(request);
    dataItemTask
            Usuario
            .addOnSuccessListener(new OnSuccessListener<DataItem>() {
                Usuario
                @Override
                public void onSuccess(DataItem dataItem) {
                    Log.d(TAG,  msg: "Sending message was successful");
                }
            })
            Usuario
            .addOnFailureListener(new OnFailureListener() {
                Usuario
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.e(TAG,  msg: "Sending message failed: " + e);
                }
            });
}
```

Figura 6.3: Método "sendData" Para la Transmisión de datos

6.3.2. Almacenamiento de los datos

Una vez se han transferido los datos se almacenan, la opción más realista seria utilizar un servidor o almacenarlos en la nube, esto implicaría el uso de servicios a terceros, google u otro tipo de almacenamiento en nube y la implementación de funcionalidades adicionales, ya que este no es el objetivo principal del proyecto y la estructura de datos es muy simple se opta por almacenar los datos de manera local en el teléfono móvil.

Se va a utilizar la librería SQLite para el almacenamiento en la app móvil, se han usado dos clases, SQLiteDatabase [27] y SQLiteOpenHelper [28].

La clase principal va a extender **SQLiteOpenHelper** e implementar el método **onCreate**. Este método es responsable de crear la base de datos, si aún no existe; en caso contrario, simplemente accede a ella. **onCreate** se ejecuta cada vez que se instancia un objeto de la clase definida.

También es necesario implementar el método **onUpgrade**, que se invoca cuando hay cambios en la estructura de la base de datos. En este caso, no se contempla dicha posibilidad.

6.4. Visualización de los datos

Esta sección describe la implementación de las funcionalidades relacionadas con la visualización de datos almacenados. Para mostrar el historial mediante gráficos, se utilizó la librería MpAndroidChart [29] para la representación en el mapa se empleó el SDK de Google Maps [30].

6.4.1. MpAndroidChart

MpAndroidChart es una librería para la creación de gráficos a partir de datos, es de código abierto y creada por un usuario particular, PhilJay en github, sus principales ventajas son la capacidad de personalización que ofrece a la hora de crear gráficos, en el proyecto se usan dos tipos de gráficos, de columnas y de líneas.

Cada grafico necesitará uno o varios objetos de tipo *lineChart* o *barChart* dependiendo del tipo de grafico además de parámetros para la personalización,

Cada consulta se resuelve directamente en la base de datos dependiendo del periodo de tiempo que se buscara y los datos de cada gráfico, frecuencia cardiaca o saturación de oxígeno y se obtiene la lista de datos sin tratar.

Estos datos pasan por el método **renderData** que los prepara dependiendo de si es un gráfico semanal, diario o mensual, y crea el objeto de tipo *lineChart o barChart* después el método **setData** crea el grafico en si con los datos preparados, de esta forma se reutiliza código que de otra forma ocuparía mucho espacio.

6.4.2. Maps SDK

El historial también se mostrará en un mapa genérico de google maps, el mapa se centrará en la ubicación actual del teléfono, por cómo se almacenaron los datos, cada instancia tendrá su ubicación y tiempo, la visualización de datos de forma geográfica resulta simple, se calculará la media de todos los valores con los mismos valores de latitud/longitud teniendo en cuenta un margen de error y se mostraran en el mapa pop ups que los representaran.

Para poder utilizar el SDK de google maps se debe crear un proyecto en google cloud platform generar y copiar la clave en el build de gradle, si no el mapa no se visualizara correctamente,

6.5. Dificultades en el desarrollo

La última sección del capítulo recoge los problemas que surgieron durante el desarrollo y como se solucionaron.

- **Navegación de los menús:** Inicialmente la aplicación del reloj contaba con un proceso único, el análisis de la frecuencia cardiaca en reposo de forma periódica, el ser un proceso único conllevaba un flujo único, siempre empezaba y terminaba igual a no ser que se produjesen incidencias y no se podía retroceder en el flujo, una vez que se avanzaba al siguiente paso había que continuar hasta terminar el proceso.

  Al introducir dos nuevos procesos, el análisis en reposo para la medición de oxígeno y el análisis de forma activa, se introduce una navegación de menús para poder seleccionar el proceso deseado y se introducen los botones para poder volver y salir de un proceso o volver al menú principal

- **Concurrencia de hilos:** El problema surge de que la monitorización periódica se realiza una vez cada 30 minutos a no ser que sea parada de forma manual por el usuario, durante el desarrollo inicial se utilizaba el mismo hilo y listener para esta monitorización y la activa, esto supuso el problema de que la monitorización en segundo plano no se podría llevar a cabo junto con la monitorización activa.

  Se soluciona el problema creando un nuevo listener que solo controlara la monitorización del oxígeno en sangre “oxygenListener”

- **Cálculo de los índices máximos y mínimos de saturación de oxígeno del usuario:** El cálculo de los valores que se muestran desde la app móvil en las gráficas que representan valores anómalos, o muy altos o muy bajos, se tiene que calcular de acuerdo al género y la edad del usuario, es decir que podrá variar si el usuario decide introducir valores nuevos.

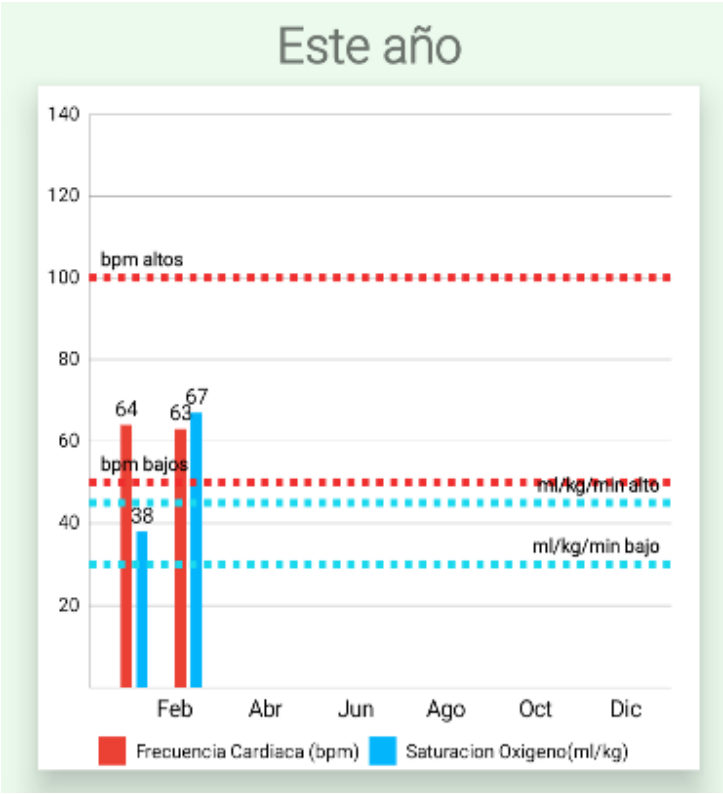
  Inicialmente se había propuesto calcular estos límites directamente en la app móvil y mostrar los valores, el problema surge de que el usuario los introduce desde el reloj, por tanto, almacenarlos de forma temporal en el reloj deja de ser una opción.

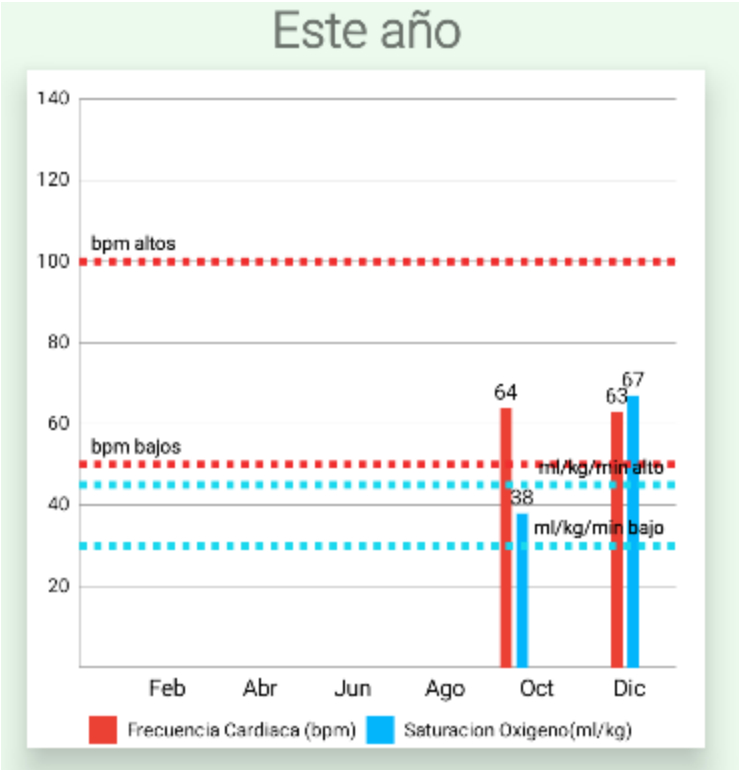
  La primera solución sería almacenar la edad y el género en la base de datos y calcular los límites, la segunda calcular los limites en el Smart Watch y almacenar los limites directamente, finalmente se optó por la segunda ya que acaba siendo la que menos datos almacena en la base de datos, por tanto, se amplía la tabla de la base de datos con dos nuevos datos, “upperLimit” y “lowerLimit”.
- **Cambio en la fórmula del cálculo de la saturación de oxigeno:** Al modificar el planteamiento de la fórmula para el cálculo de la saturación de oxígeno [6.1]y introducir en la formula factores como edad, el género y peso el cálculo requiere de más tiempo de actividad por parte del usuario, esto se corrige implementando un tiempo mínimo en el que se requiere al usuario mantener

actividad constante, una vez el periodo concluye se inicia el análisis, cuanto más largo el periodo, más fiables serán los resultados.

- **Generación de gráficos de barras vacíos:** A la hora de calcular el valor promedio de los gráficos de barras anual y añadirlo a la gráfica, se generaba la gráfica sin dejar espacio para los meses para los que no existía un valor.

  Inicialmente se consideraba este error como simple formato de las gráficas en sí, después de muchos dolores de cabeza se descubrió que la gráfica solo tenía en cuenta los valores si se introducían vacíos (x:0 y:0) e inicialmente se estaban intentando meter solo los meses de los cuales existían valores por cómo se estaban procesando los datos.





Comparación de antes y después con gráficos con los mismos datos

Capítulo 7

Pruebas

Las pruebas son una parte fundamental de cualquier proyecto de software. En proyectos más grandes, que implican la colaboración de múltiples desarrolladores y una considerable inversión de tiempo y recursos, es imprescindible realizar diversas pruebas durante el desarrollo. A medida que el tamaño del código aumenta, los errores se vuelven más difíciles de identificar, lo que les da aún más importancia a las pruebas para garantizar la calidad y el correcto funcionamiento del software.

Pruebas unitarias: Su objetivo es verificar que cada unidad o componente individual del software funcione de manera correcta. Estas pruebas se realizan de forma aislada y constituyen la primera etapa del proceso de pruebas durante el desarrollo.

Pruebas de integración: Estas pruebas tienen como propósito garantizar que los distintos componentes del software operen correctamente cuando interactúan entre ellos. Es común que un módulo que opera perfectamente de manera individual genere errores al integrarse con otros. Por ello, es fundamental supervisar el proceso de integración para asegurar el correcto funcionamiento del sistema. Al ser tests más complejos, se realizan después de completar las pruebas unitarias.

Pruebas funcionales: Estas pruebas evalúan las funcionalidades del software para determinar su usabilidad y adecuación a requisitos del mercado. Se clasifican como pruebas de caja negra, ya que se centran en verificar el comportamiento del sistema sin considerar su estructura interna. El objetivo principal es comprobar que el software cumple con lo especificado en el documento de requisitos.

Pruebas de aceptación: Antes de iniciar el proyecto, el equipo responsable debe establecer los criterios de aceptación que definirán el éxito del software. Además, cualquier cambio en estos criterios durante el desarrollo, ya sea para añadir o eliminar requisitos, debe ser documentado adecuadamente. Estas pruebas se llevan a cabo, tanto durante las pruebas unitarias como en las de integración, para garantizar que el sistema en su conjunto funcione conforme a lo esperado y cumpla con los objetivos definidos.

Pruebas de rendimiento: Su objetivo es evaluar cómo responde el software bajo diferentes cargas de trabajo y en condiciones reales de uso. Estas pruebas permiten medir aspectos clave como la estabilidad y la velocidad, factores directamente relacionados con la experiencia del usuario y la tasa de conversión de la aplicación.

Pruebas de estrés: Antes de finalizar el desarrollo, es fundamental verificar cuánto puede soportar el sistema antes de que se produzcan errores. Estas pruebas consisten en someter al software a una carga de información significativamente mayor a la habitual, con el propósito de identificar el punto en el que se satura o falla.

Pruebas de regresión: Cuando se realizan modificaciones en una funcionalidad del sistema, estas pruebas se encargan de garantizar que los cambios no generen errores inesperados o afecten negativamente a otros componentes no modificados. Son esenciales para asegurar que la incorporación de nuevas funciones no comprometa el correcto funcionamiento del resto del sistema.

Pruebas de humo: Estas pruebas funcionales tienen como finalidad determinar si el software completo opera correctamente y está listo para someterse a evaluaciones más exhaustivas. Su objetivo principal es validar las funcionalidades esenciales; si no se superan, no se avanza hacia pruebas más complejas.

Tras esta introducción sobre las pruebas más comunes que se realizan en un proyecto de software en un entorno profesional, esta memoria se centrará en las pruebas unitarias de algunas funcionalidades específicas del sistema.

7.1. Pruebas unitarias: Unit Testing

Las pruebas unitarias del proyecto se realizaran en Android Studio con JUnit [31], una extensión compuesta por varias bibliotecas que facilitan las pruebas unitarias en aplicaciones Java.

El framework JUnit permite ejecutar clases de java de tipo test de forma aislada, cubriendo solo la clase a la que corresponde el test, los unit test funcionan de la siguiente forma.

Se debe determinar el comportamiento esperado por la clase en condiciones específicas, este comportamiento “ideal” se llama aserto, después el test compara el comportamiento real con el aserto, si el comportamiento es el mismo el test se considera exitoso, si devuelve un valor diferente el test dará un error, obviamente estas pruebas van a estar condicionadas por el input de la persona probándola, estos test tienen más sentido cuando se trata de un proyecto multitudinario.

Los test [32] se pueden ejecutar de forma individual y no requieren que la aplicación este en ejecución en la mayoría de los casos, una vez completado aparecerá un tick verde indicando que el test ha sido correcto o una cruz roja junto con su error de ejecución.

7.1.1. Indicar estado de reposo en análisis activo

En el supuesto de este test se está realizando el análisis activo, para el que se requiere actividad física constante, en el caso de que el usuario entre en reposo se modificara la interfaz con el método *updateNotRest*, en el test se crean dos *textViews* con el método *mock* un método único de las pruebas unitarias que creara una copia del objeto que referencia para no interferir con el código real y mantener la ejecución del test aislada, después se instancia la clase real *updateFragment* y se compara el resultado simulado con el esperado, con el método *verify* también especifico de unit testing.

```
@Test
void updateNotActiveTest() {

    TextView mockRestIndicator = mock(TextView.class);
    TextView mockHeartRate = mock(TextView.class);

    ResumeFragment resumeFragment = new ResumeFragment();
    resumeFragment._restIndicator = mockRestIndicator;
    resumeFragment._heartRate = mockHeartRate;

    String expectedRestIndicatorText = ("No estas activo," +
            " se requiere actividad fisica para el analisis");
    String expectedHeartRateText = "--";

    resumeFragment.updateNotActive();

    verify(mockRestIndicator).setText(expectedRestIndicatorText);
    verify(mockRestIndicator).setTextSize(20);
    verify(mockHeartRate).setText(expectedHeartRateText);
}
```

Figura 7.1: Patrón Observer

7.1.2. Detener el análisis de saturación de oxígeno en actividad

En el caso de que se detenga cualquier análisis, por falta de requerimientos o por una pausa manual del usuario, habrá que parar la ejecución de los Listeners que controlan los hilos de los sensores, tanto la frecuencia como la ubicación o el acelerómetro, de esto se encarga el método *stopListeners*, este método detendrá la ejecución de todos los Listeners activos en ese momento y se va a llamar en múltiples instancias desde varias clases, en este caso se prueba desde el listener responsable del análisis del oxígeno *OxygenListener*.

```
@Test
public void stopListeners() throws InterruptedException{

    OxygenListener OxygenListener = new OxygenListener();
    SensorManager sensorManager = mock(SensorManager.class);
    SensorEventListener eventListener = mock(SensorEventListener.class);
    Sensor accelerometerSensor = mock(Sensor.class);
    OxygenListener.setSensorManager(sensorManager);
    OxygenListener.setEventListener(eventListener);
    OxygenListener.setAccelerometerSensor(accelerometerSensor);

    Thread thread = new Thread(OxygenListener);
    thread.start();
    OxygenListener.stopListeners();

    verify(sensorManager).unregisterListener(eventListener, accelerometerSensor);
    assertFalse(OxygenListener.getIsResting());
    assertEquals( expected: 0, OxygenListener.getRestAccumulatedTime());
    assertEquals( expected: 0, OxygenListener.getRestStartTime());
    SharedData singleton = SharedData.getInstance();
    assertEquals( expected: 0, singleton.getAcceleration());

}
```

8. Conclusiones

La sección de conclusiones constituye el cierre del trabajo de fin de grado y tiene como propósito reflejar en los objetivos conseguidos a lo largo del desarrollo del proyecto. A través de este apartado se pretende ofrecer una visión global de los objetivos planteados, los resultados alcanzados y el grado en que se han cumplido las metas propuestas. Asimismo, se incluyen reflexiones sobre el proceso llevado a cabo, las limitaciones encontradas y las posibles líneas de mejora o de investigación futura que podrían surgir de este estudio.

Antes de comenzar el proyecto no había trabajado con Android en el contexto de desarrollo de aplicaciones móviles y Wear OS, comenzar con el desarrollo en si del proyecto supuesto un gran reto y la etapa inicial supuso más tiempo del planificado y dio la sensación de bloqueo inicial, una vez superada el primer sprint que consistía en codificar una aplicación de prueba con funciones básicas, el proyecto comenzó a avanzar a un ritmo más consistente.

Personalmente diría que los obstáculos superados a lo largo del proyecto han ayudado a aprender a desarrollar en el contexto de aplicaciones móviles y los conocimientos adquiridos son suficientes para si fuese necesario aplicarlos a oportunidades laborales y me sentiría cómodo incluyéndolo en mi curriculum o una entrevista de trabajo.

Respecto a los objetivos planteados, todos ellos a grandes rasgos terminan siendo, desarrollar dos aplicaciones funcionales que cumplan los requisitos propuestos, que se cumplen, la monitorización de oxigeno durante actividad física, la transmisión de datos entre las dos aplicaciones la determinación de valores personalizados para cada usuario basándonos en parámetros proporcionados por el mismo y la visualización de los datos.

8.1. Líneas de trabajo futuras

En este apartado se detallan una serie de funcionalidades que podrían implementarse para mejorar las aplicaciones actuales. Algunas de ellas son completamente nuevas, mientras que otras representan mejoras significativas de las características ya implementadas.

Análisis de Datos

Esta funcionalidad añadiría un valor significativo tanto para el mercado como para la experiencia del usuario. Por ejemplo:

> **Algoritmos de Machine Learning:** Se podrían analizar las frecuencias cardíacas para detectar posibles patologías cardiovasculares, como arritmias o analizar patrones en la saturación de oxígeno para la detección de enfermedades respiratorias
>
> **Mapa de Calor:** Una visualización geográfica de las zonas donde se registran frecuencias cardíacas elevadas en reposo podría identificar lugares asociados a mayor estrés, como el trabajo o el hogar.
> Estas aplicaciones permitirían enriquecer la utilidad de las apps y abrir nuevos campos de análisis y personalización.

Aviso a Emergencias

Esta funcionalidad integraría un acceso directo al número de emergencias desde la aplicación. En combinación con un análisis prolongado de los datos, podría recomendar al usuario que contactara a emergencias si se detectan anomalías.

Sistema de Login

Una mejora importante sería implementar un sistema de registro y autenticación de usuarios. Esto permitiría a cada usuario acceder exclusivamente a los datos de su cuenta, ofreciendo mayor privacidad y personalización.

Almacenamiento en un Servidor Externo

Actualmente, los datos se almacenan en una base de datos local. Una mejora sencilla pero efectiva sería migrar el almacenamiento a un servidor externo. Esto aseguraría la persistencia de los datos, incluso si el usuario desinstala la app o borra los datos locales. Al reinstalar la aplicación, toda la información seguiría estando disponible.

Monitorización de Presión Arterial

Algunos dispositivos smartwatch son capaces de medir este parámetro, lo que ampliaría aún más el alcance de la aplicación, aunque al igual que con el sensor de saturación e oxigeno actualmente, las APIs para estos indicadores no son tan accesibles ni estándar como las de frecuencia cardíaca, ya que muchas marcas desarrollan sus propios SDK, que suelen ser de uso restringido. No obstante, es probable que en el futuro se simplifique el acceso a estos sensores, facilitando su integración en las apps.

III

Manuales de la Aplicación

Pablo Polo Mínguez

Capítulo 9

Manual de Instalación

Instalación de las Aplicaciones

En este capítulo se explicará cómo instalar ambas aplicaciones desde cero. Se describirán los pasos necesarios para configurar el entorno de desarrollo y clonar el repositorio del proyecto.

9.1 Instalación del Entorno y Clonado del Repositorio

Para instalar las aplicaciones, es imprescindible contar con **Android Studio**, el IDE utilizado durante todo el desarrollo del proyecto. Este será el entorno donde se importará y ejecutará el proyecto. A continuación, se detallan los requisitos mínimos del sistema según el sistema operativo:

Windows:

- **Sistema Operativo:** Microsoft Windows 7/8/10 (32-bit o 64-bit).
- **Memoria RAM:** 3 GB mínimo (recomendados 8 GB, más 1 GB adicional para el emulador de Android).
- **Espacio en Disco:** 2 GB mínimo (recomendados 4 GB: 500 MB para el IDE y 1.5 GB para el SDK de Android y la imagen del sistema del emulador).

Mac OS:

- **Sistema Operativo:** Mac OS X 10.10 (Yosemite) o superior, hasta 10.13 (High Sierra).
- **Memoria RAM:** 3 GB mínimo (recomendados 8 GB, más 1 GB adicional para el emulador de Android).
- **Espacio en Disco:** 2 GB mínimo (recomendados 4 GB: 500 MB para el IDE y 1.5 GB para el SDK de Android y la imagen del sistema del emulador).

Linux:

- **Sistema Operativo:** Una distribución moderna de 64 bits, como Ubuntu 18.04 o superior.
- **Bibliotecas Requeridas:** glibc 2.19 o superior, y libstdc++.
- **Memoria RAM:** 3 GB mínimo (se recomiendan 8 GB, más 1 GB adicional para el emulador de Android).
- **Espacio en Disco:** 2 GB mínimo (se recomiendan 4 GB: 500 MB para el IDE y 1.5 GB para el SDK de Android y la imagen del sistema del emulador).

Pasos para Instalar el Entorno

Descargar Android Studio:

Ir al sitio web oficial de Android Studio.

Descargar la versión adecuada para tu sistema operativo, ejecutar el instalador y seguir las instrucciones del asistente de instalación.

Asegurarse de instalar el SDK de Android, las herramientas necesarias, y el emulador de Android si es necesario.

Clonar el Repositorio del Proyecto:

Abrir una terminal o línea de comandos.

Navegar a la carpeta donde deseas clonar el proyecto.

Ejecutar el siguiente comando para clonar el repositorio:

bash

CopyEdit

git clone <URL_del_repositorio>

Reemplazar <URL_del_repositorio> con la URL del repositorio del proyecto.

Importar el Proyecto en Android Studio:

Abrir Android Studio y seleccionar **"Open or Import Project"**.

Navegar a la carpeta donde se clonó el repositorio y seleccionarlo.

Seguir las indicaciones para sincronizar el proyecto con Gradle.

9.2. Instalación de la app del teléfono

Para la instalación de la aplicación del teléfono se requiere una versión igual o superior a 8.0 de Android y al menos 20 Mb de almacenamiento.

La instalación de la aplicación se realizará mediante la depuración por usb, para esto habrá que permitir su uso desde la configuración del propio teléfono.

Activar las Opciones de Desarrollador

Si no aparecen las opciones de desarrollador normalmente es porque no están activadas:

1. **Abrir Configuración:** Navega a la configuración del teléfono.
2. **Ir a "Acerca del Teléfono":** Busca y selecciona esta opción.
3. **Desbloquear las Opciones de Desarrollador:** Pulsa siete veces en la opción **"Número de compilación"**, aparecerá un mensaje indicando que las opciones de desarrollador han sido desbloqueadas.

Habilitar la Depuración por USB

4. **Abrir las Opciones de Desarrollador:** Ve a Configuración y selecciona **"Opciones de desarrollador"**.
5. **Activar la Depuración por USB:** Busca y activa esta opción.

Instalar y Ejecutar la App desde Android Studio

1. **Conectar el Smartphone al Ordenador:** Usa un cable USB compatible.
2. **Abrir el Proyecto en Android Studio:** Inicia Android Studio y abre el proyecto correspondiente.
3. **Seleccionar el Dispositivo:**
    - Android Studio reconocerá automáticamente el dispositivo conectado.
    - En la barra superior, selecciona:
        - El módulo del proyecto
        - Tu dispositivo.
4. **Ejecutar la App:**
    - Haz clic en el botón **"Run"**
    - Esto instalará y ejecutará la app directamente en tu smartphone.

Notas Adicionales

- Si Android Studio no reconoce el dispositivo
    - Asegúrate de que los controladores USB necesarios están instalados en el ordenador.
    - Comprueba que el teléfono está configurado para **"Transferencia de archivos"** en lugar de solo carga.
- Una vez instalada, la app se podrá ejecutar directamente desde el teléfono sin necesidad de conexión al ordenador.

9.3. Instalación de la app del reloj

Para instalar la aplicación en el reloj se requerirá un dispositivo compatible con Android WearOS y un ordenador conectado a la misma red wifi que el reloj.

La instalación es similar a la app móvil excepto que en vez de la depuración usb se usara la depuración por wifi.

Activar las Opciones de Desarrollador:

1. Ir a **Configuración > Acerca del reloj**.
2. Pulsar **siete veces** sobre **"Número de compilación"** hasta que se activen las opciones de desarrollador.

Activar la Depuración por Wifi:

1. Ir a **Opciones de desarrollador**.
2. Activar **Depuración ADB** y **Depuración por Wifi**.

Conectar el Ordenador al Reloj

1. **Obtener la dirección IP del reloj:**
    - Ir a **Configuración > WiFi** en el reloj.
    - Pulsar sobre la red a la que está conectado.
    - Copiar la **dirección IP** que aparece.
2. **Abrir el terminal en Android Studio:**
    - Ir a la pestaña **Terminal** dentro de Android Studio.
3. **Navegar hasta la carpeta "platform-tools":**

   Introducir el siguiente comando en el terminal (reemplazando <usuario> con el nombre de usuario en el PC):

   ```
   cd \Users\<usuario>\AppData\Local\Android\Sdk\platform-tools
   ```

   (En macOS o Linux, la ruta suele estar en *~/Android/Sdk/platform-tools/*).

   Conectar el reloj al ordenador con ADB:

   - Ejecutar el siguiente comando, reemplazando <*direccion_IP*> con la dirección copiada del reloj:

     ```
     adb connect <direccion_IP>
     ```

     La dirección ip del reloj se puede visualizar en el apartado de depuración inalámbrica del reloj, tendrá un formato similar a 192.168.1.130

   - Si la conexión es exitosa, aparecerá un mensaje confirmándolo.

10.Manual de Usuario

Esta sección funciona como un breve tutorial o guía para que el usuario se familiarice con las aplicaciones y su funcionamiento, además de poder visualizar las interfaces en el contexto de la memoria del trabajo.

10.1. Manual de la app del smartwatch

Una vez se inicializa la aplicación por primera vez se muestra el menú principal para que el usuario elija la actividad que quiere realizar o la medición que quiere realizar.

10.1.1 Análisis de la frecuencia cardiaca en reposo.

Si se elige el análisis de la frecuencia cardiaca en reposo el análisis se inicia inmediatamente, la primera parte del análisis, será la monitorización de la actividad, requiriendo al usuario que se mantenga en reposo mientras pasa la cuenta atrás, si se detecta actividad se reiniciara la cuenta atrás.

Una vez completado el periodo de reposo se comienzan a leer valores de la frecuencia cardiaca, actualizándose en tiempo real también con una cuenta atrás para que el usuario sepa el tiempo restante

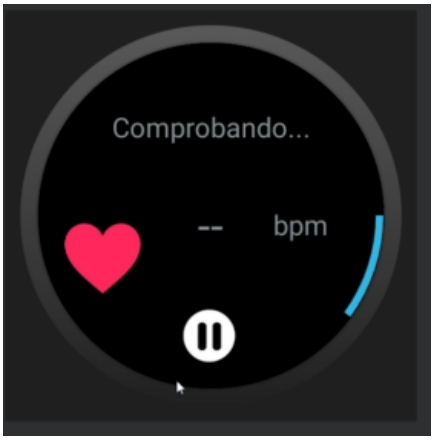
Cuando finaliza el análisis se muestra al usuario el valor final y un botón para regresar al menú principal que muestra un mensaje indicando que se ha finalizado el análisis.

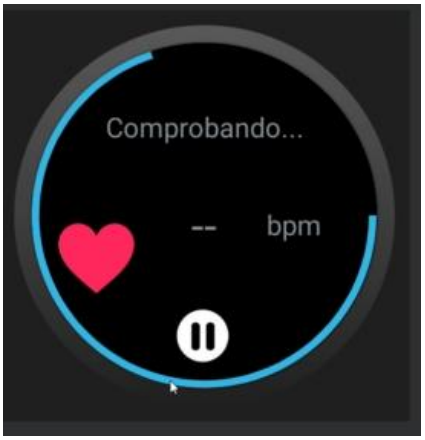
10.1.2 Análisis del oxígeno en sangre en actividad

Al seleccionar esta opción se mostrará una ventana para pedir al usuario los datos necesarios para este análisis, su edad, peso y género, con la posibilidad de volver al menú anterior para cambiar los valores.

Una vez los 3 valores están en la memoria del reloj este menú no se mostrará, se muestra una opción para iniciar el análisis o volver a introducir de nuevo los datos.

Cuando empieza la monitorización se exige un periodo de actividad al usuario en el que se muestra una barra azul para designar progreso, si se detecta reposo se mostrara un mensaje por pantalla hasta que se vuelva a retomar la actividad.





Una vez completada la barra de progreso se comienzan a analizar los valores, el usuario podrá estar en reposo, se muestra una cuenta atrás para saber cuándo finaliza en análisis y se muestra el valor final al usuario.

10.2. Manual de la app del móvil

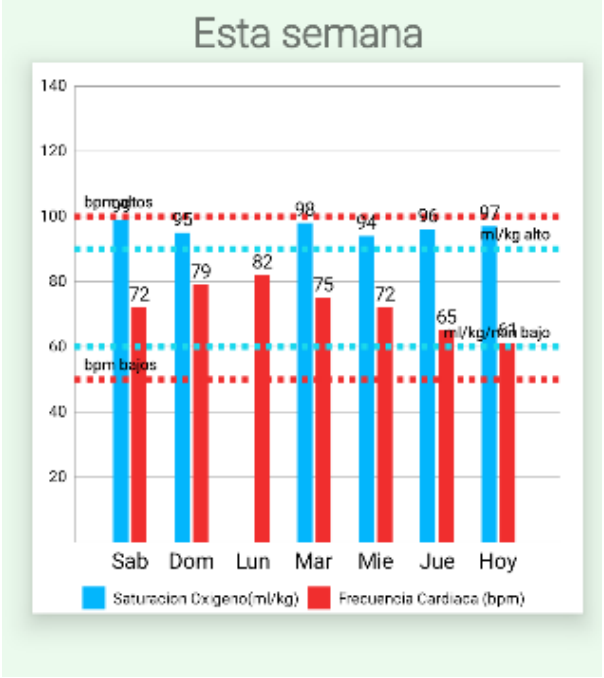
Utilizando la barra de navegación inferior se navega entre las pestañas principales

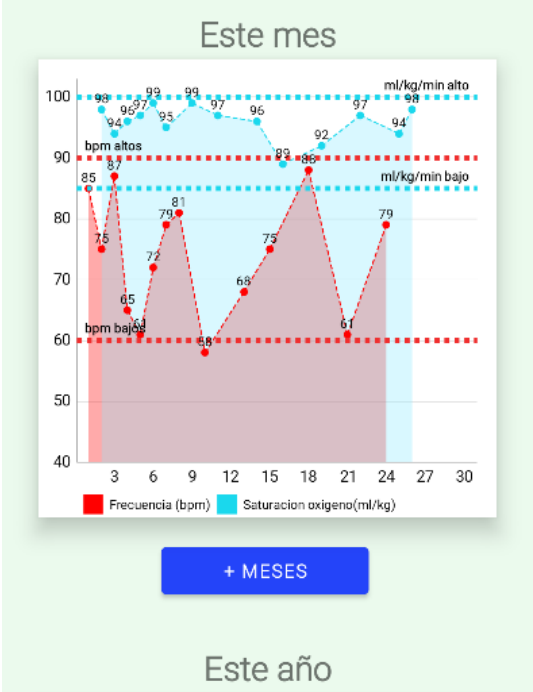
10.2.1 Pantalla de Inicio

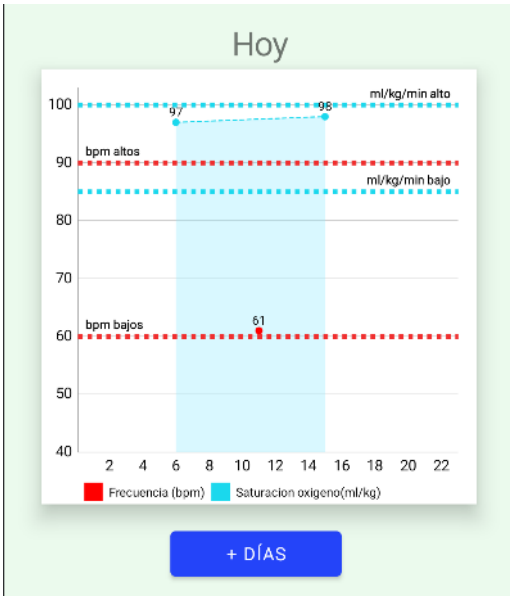
Al iniciar la app del Smartphone se muestra inicialmente un resumen de la última medición realizada con la fecha y la ubicación.

10.2.2 Pestaña de visualización de valores o historial

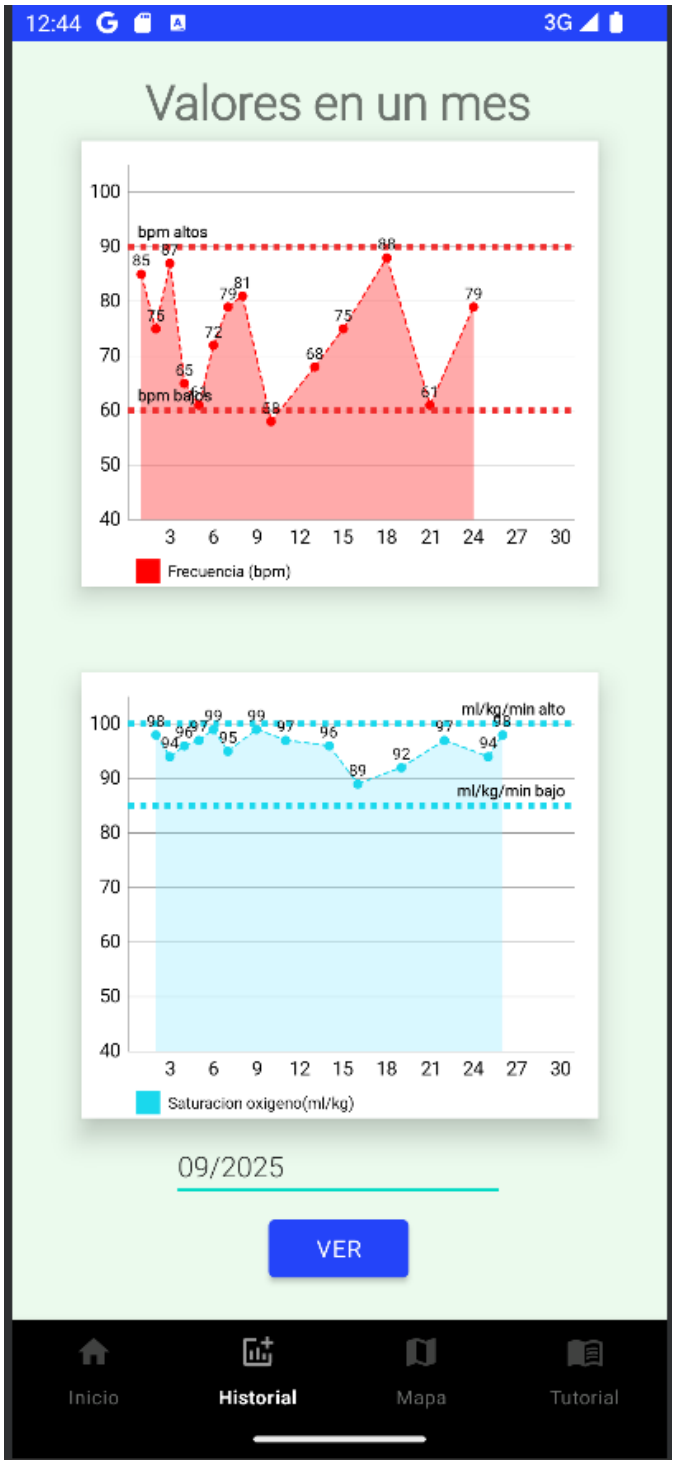
En esta pestaña se visualizan todos los gráficos de los valores con un gráfico por cada franja de tiempo, última semana, mes, año y el día actual por horas con dos tipos de gráficos, barras y puntos.

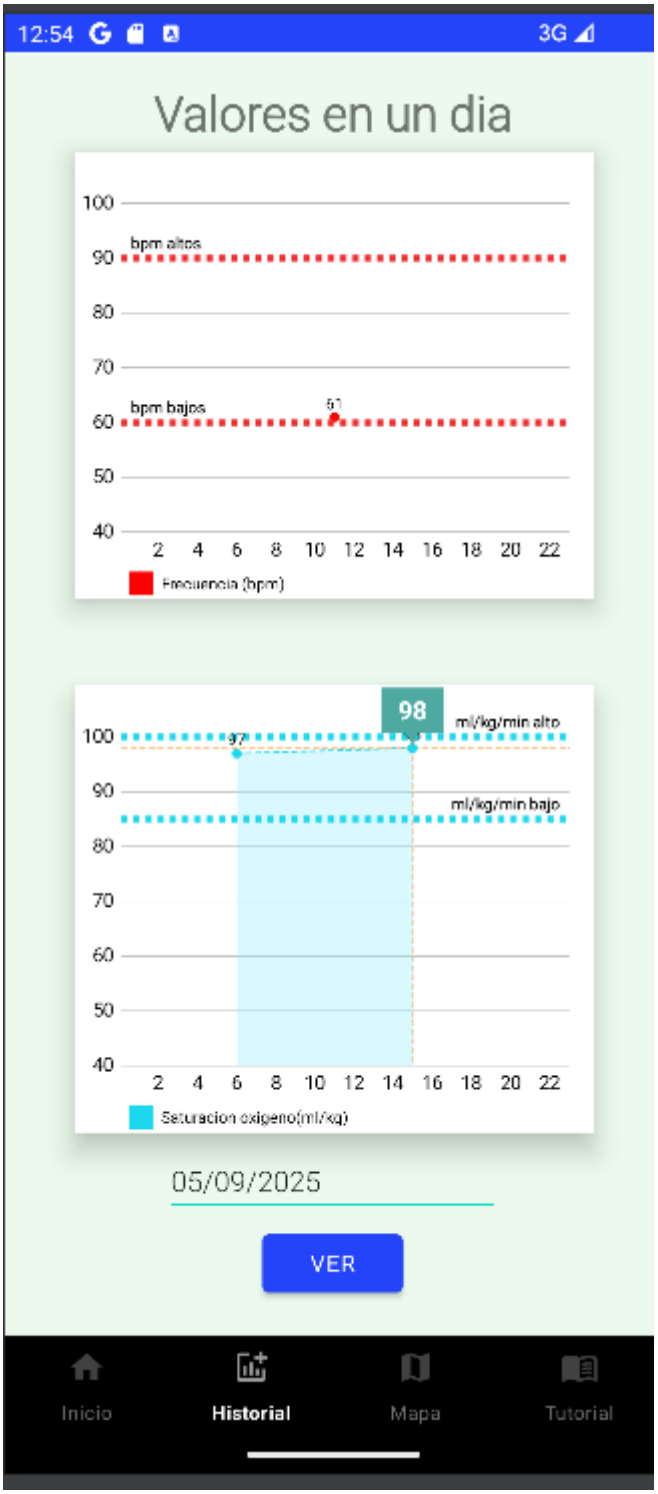






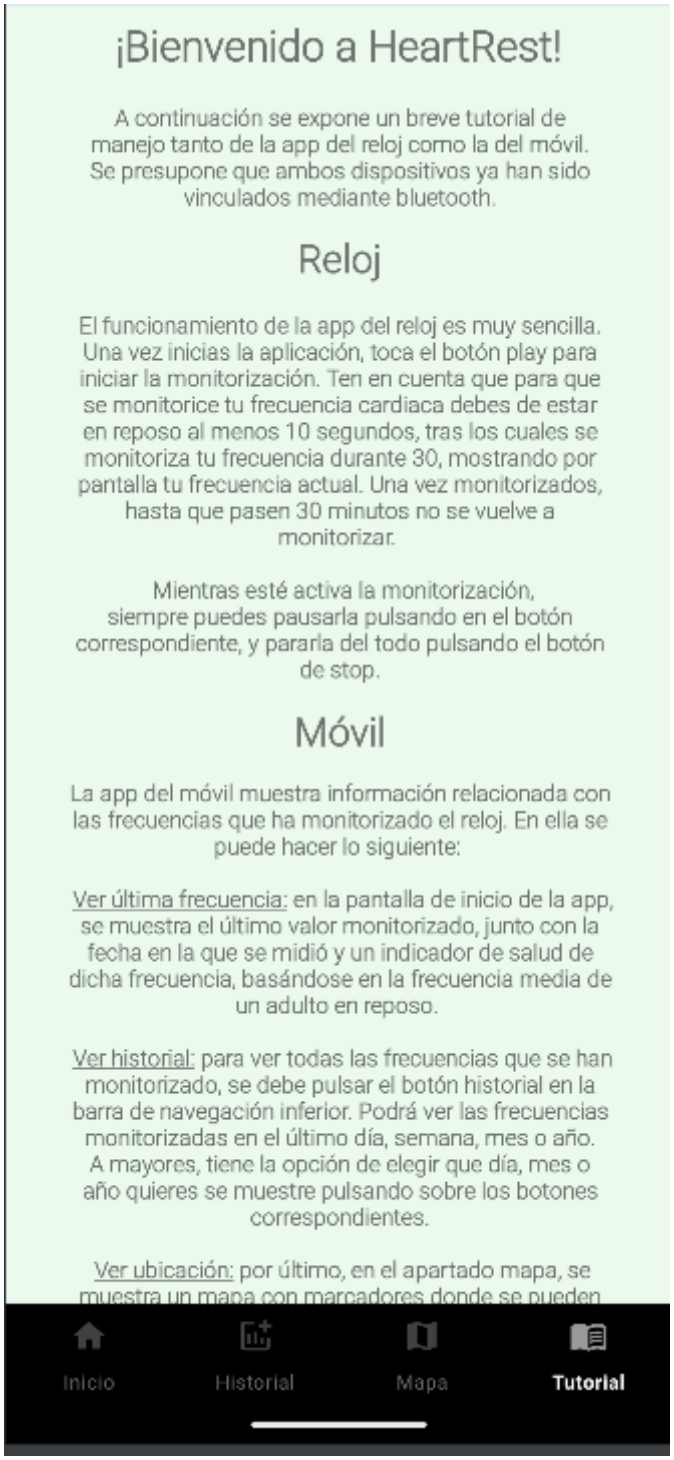
En la vista general se muestran los dos valores en el mismo gráfico simultáneamente, como esta vista puede ser un tanto ocupada se puede seleccionar +meses +días o +años para obtener una vista detallada con los dos gráficos por separado.





Pablo Polo Mínguez

10.2.3 Pestaña del tutorial



Webgrafía

[1] Samsung Developers. [Online]. Última vez visitado: 18/07/2024. URL: https://developer.samsung.com/galaxy-watch

[2] Draw.io [Online]. Última vez visitado: 03/11/2024. URL: https://app.diagrams.net/

[3] Samsung Developer Forum. [Online]. Última vez visitado: 05/08/2024. URL: https://forum.developer.samsung.com/t/galaxy-watch-4-sensor-data/16454

[4] Stack Overflow. [Online]. Última vez visitado: 22/05/2024. URL: https://stackoverflow.com/questions/75764660/wear-os-3-read-data-from-blood-oxygen-saturation-sensor

[5] Google Developers - Fit API. [Online]. Última vez visitado: 09/06/2024. URL: https://developers.google.com/fit/android/sensors?hl=es-419

[6] Android Developers - Principios de diseño en Wear OS. [Online]. Última vez visitado: 28/09/2024. URL: https://developer.android.com/training/wearables/principles?hl=es-419

[7] JetBrains - IntelliJ IDEA. [Online]. Última vez visitado: 10/8/2024. URL: https://www.jetbrains.com/idea/

[8] Cables & Sensors. [Online]. Última vez visitado: 27/10/2024. URL: https://www.cablesandsensors.com/pages/difference-between-spo2-and-vo2max

[9] Stack Overflow - adb connect fix. [Online]. Última vez visitado: 03/10/2024. URL: https://xdaforums.com/t/adb-connect-over-network-is-refused.4564381/

[10] Stack Overflow - Esconder y enseñar imágenes. [Online]. Última vez visitado: 25/08/2024. URL: https://stackoverflow.com/questions/13397709/android-hide-imageview

[11] Stack Overflow - Modificar imagen dinámicamente. [Online]. Última vez visitado: 27/08/2024. URL: https://stackoverflow.com/questions/2974862/changing-imageview-source

[12] Formulas VO2. [Online]. Última vez visitado: 15/09/2024. URL: https://www.mdapp.co/vo2-max-calculator-for-aerobic-capacity-369/

[13] Tablas de valores VO2. [Online]. Última vez visitado: 20/10/2024. https://www.babelsport.com/blog/consejos-running/v02-maximo-tablas-edad-sexo/

[14] Android Developers - Picker Tutorial. [Online]. Última vez visitado: 12/09/2024. URL: https://www.youtube.com/watch?v=UMc7Tu0nKYQ&ab_channel=AndroidDevelopers

[15] Stack Overflow - Superposición de gráficos de barras. [Online]. Última vez visitado: 30/09/2024. URL: https://stackoverflow.com/questions/47246497/how-to-overlap-two-barcharts-over-each-other

[16] Android Developers - Progress Indicator. [Online]. Última vez visitado: 18/07/2024. URL: https://developer.android.com/design/ui/wear/guides/components/progress-indicator#adaptive-layouts

[17] Stack Overflow - Uso de useLevel en Shape. [Online]. Última vez visitado: 08/11/2024. URL: https://stackoverflow.com/questions/35803975/android-purpose-of-uselevel-in-shape-tag

[18] Glassdoor. [Online]. Última vez visitado: 15/07/2024. URL: https://www.glassdoor.es/Sueldos/android-developer-sueldo-SRCH_KO0,17.html

[19] Tarifa Luz Hora. [Online]. Última vez visitado: 15/07/2024. URL: https://tarifaluzhora.es/

[20] Wikipedia - Patrón de diseño. [Online]. Última vez visitado: 22/09/2024. URL: https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o

[21] Google Developers - Fused Location Provider. [Online]. Última vez visitado: 05/08/2024. URL: https://developers.google.com/location-context/fused-location-provider

[22] Android Developers - LocationManager. [Online]. Última vez visitado: 30/08/2024. URL: https://developer.android.com/reference/android/location/LocationManager

[23] Android Developers - Bound Services. [Online]. Última vez visitado: 12/10/2024. URL: https://developer.android.com/develop/background-work/services/bound-services?hl=es-419

[24] Android Developers - Handler. [Online]. Última vez visitado: 07/11/2024. URL: https://developer.android.com/reference/android/os/Handler

[25] Android Developers - Data Sync en Wear OS. [Online]. Última vez visitado: 18/06/2024. URL: https://developer.android.com/training/wearables/data/overview

[26] Google Developers - PutDataMapRequest. [Online]. Última vez visitado: 23/07/2024. URL: https://developers.google.com/android/reference/com/google/android/gms/wearable/PutDataMapRequest

[27] Android Developers - SQLiteDatabase. [Online]. Última vez visitado: 29/08/2024. URL: https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase

[28] Android Developers - SQLiteOpenHelper. [Online]. Última vez visitado: 14/09/2024. URL: https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper

[29] GitHub - MPAndroidChart. [Online]. Última vez visitado: 06/12/2024. URL: https://github.com/PhilJay/MPAndroidChart

[30] Google Developers - Maps SDK para Android. [Online]. Última vez visitado: 20/08/2024. URL: https://developers.google.com/maps/documentation/android-sdk/start?hl=es-419

[31] Android Developers - Pruebas locales en Android. [Online]. Última vez visitado: 28/10/2024. URL: https://developer.android.com/training/testing/local-tests

[32] Stack Overflow - Creación de tests en Android Studio. [Online]. Última vez visitado: 08/11/2024. URL: https://stackoverflow.com/questions/16586409/how-can-i-create-tests-in-android-studio

[33] Android Developers - Android Basics Compose (Unidad 4). [Online]. Última vez visitado: 15/08/2024. URL: https://developer.android.com/courses/pathways/android-basics-compose-unit-4-pathway-1?hl=es-419

[37] Reddit - Wear OS Dev. [Online]. Última vez visitado: 12/06/2024. URL: https://www.reddit.com/r/WearOSDev/comments/qgpy3s/api_level_for_galaxy_watch_4/

[38] Vo2 tests [Online]. Última vez visitado: 14/09/2024. URL: https://www.youtube.com/watch?v=s1sMgQXbuRs&ab_channel=TheMovementSystem

[39] Stack Overflow - invalidate(). [Online]. Última vez visitado: 05/07/2024. https://stackoverflow.com/questions/37228741/custom-view-class-ondraw-isnt-called-by-invalidate

[40] Sdk de oxigeno de Samsung. [Online]. Última vez visitado: 24/11/2024 https://developer.samsung.com/codelab/health/blood-oxygen.html#Overview

[41] Api rest de Android. [Online]. Ultima vez visitado: 17/08/2024 https://developers.google.com/fit/android/sensors?hl=es-419

[42]: Desarrollo iteractivo. [Online]. Ultima vez visitado: 24/09/2024

https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente

[42]: Ventajas de usar metodológica Agile. [Online]. Ultima vez visitado: 24/09/2024

https://creatiabusiness.com/9-ventajas-de-usar-la-metodologia-agile/

[43]: Metodologías Agile vs Waterfall: [Online]. Ultima vez visitado: 24/09/2024 https://www.seguetech.com/waterfall-vs-agile-methodology/

[44]: Arquitectura Android: [Online]. Ultima vez visitado: 17/09/2024

https://developer.android.com/topic/architecture?hl=es-419

[45] Samsung Health sensor SDK overview [Online] Ultima vez visitado: 17/09/2024

https://developer.samsung.com/codelab/health/skin-temperature.html#Overview

[46] Patrón Observer [Online] Ultima vez visitado: 20/09/2024

https://gustavopeiretti.com/patron-de-diseno-observer-en-java/

[47] Samsung Health SDK overview [Online] Ultima vez visitado: 17/09/2024

https://columna80.wordpress.com/2011/02/17/arquitectura-de-android/

[48] Apple Watch Series 10 [Online] Última vez visitado: 08/09/2025
https://www.apple.com/apple-watch-series-10/specs/

[49] Apple Watch Series 10 Health insights [Online] Última vez visitado: 08/09/2025
https://www.apple.com/apple-watch-series-10/

[50] Galaxy Watch7 funciones de salud y sensores [Online] Última vez visitado: 08/09/2025
https://www.samsung.com/us/watches/galaxy-watch7/

[51] Galaxy Watch7 vs Watch6 BioActive Sensor y SpO_2 [Online] Última vez visitado: 08/09/2025
https://www.samsung.com/us/support/answer/ANS10003275/

[52] Huawei Watch GT 4 especificaciones SpO_2 [Online] Última vez visitado: 08/09/2025
https://consumer.huawei.com/en/wearables/watch-gt4/specs/

[53] Huawei Watch GT4 detección automática de SpO_2, monitorización continua [Online] Última vez visitado: 08/09/2025
https://www.watchesonline.com/huawei-watch-gt4-46mm-active-edition-black

[54] Wearable Heart Beat Register-Apps: Registro automatizado de la frecuencia cardiaca en reposo a través de un smartwatch [Online] Última vez visitado: 08/09/2025
https://uvadoc.uva.es/handle/10324/59650

[55] Grafico de metodología agile [Online] Última vez visitado: 08/09/2025
https://www.inesdi.com/sites/default/files/inline-images/metodo-scrum_0.jpg