Joel Panisello Lozano

DESARROLLO DE APLICACIÓN “DRIVING ANGEL” PARA LA DETECCIÓN DE SOMNOLENCIA AL VOLANTE

TRABAJO DE FIN DE GRADO

Dirigido por Sr.Dr. Carlos Barberà Escoí

Grado de Ingeniería Informática

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2023

Resumen

Una parte de los accidentes en carretera son debidos al cansancio. Para intentar evitarlos, los profesionales del transporte de camiones tienen que hacer paradas cada cierto tiempo, pero hay muchos otros conductores que no están sometidos a la vigilancia del tacógrafo.

El proyecto consiste en diseñar una aplicación móvil que reconozca el tiempo que lleva el usuario conduciendo y que le haga recomendaciones para parar a descansar.

La aplicación utilizará un smartwatch con Android Wear OS[1] para vigilar tanto la actividad del conductor (conduciendo o descansando) así como el ritmo cardíaco para evitar accidentes avisando al conductor mediante una alarma sonora.

Palabras Clave: aplicación móvil, relojes inteligentes, detección de somnolencia.

Resum

Una part dels accidents en carretera són deguts al cansament. Per intentar evitar-los, els professionals del transport de camions han de fer parades cada cert temps, però hi ha molts altres conductors que no estan sotmesos a la vigilància del tacògraf.

El projecte consisteix a dissenyar una aplicació mòbil que reconegui el temps que porta l'usuari conduint i que li faci recomanacions per parar a descansar.

L'aplicació utilitzarà un smartwatch amb Android Wear OS per vigilar tant l'activitat del conductor (conduint o descansant) així com el ritme cardíac per evitar accidents avisant el conductor mitjançant una alarma sonora.

Paraules Clau: aplicació mòbil, rellotges intel·ligents, detecció de somnolència.

Abstract

A proportion of road accidents are due to fatigue. To try to avoid them, truck transport professionals must stop every so often, but there are many other drivers who are not subject to tachograph monitoring.

The project consists of designing a mobile application that recognises how long the user has been driving and makes recommendations to stop and rest.

The application will use a smartwatch with Android Wear OS to monitor both the driver's activity (driving or resting) and heart rate to prevent accidents by warning the driver with an audible alarm.

Key Words: mobile application, smartwatches, detection of drowsiness.

[1] Operative System

Índice

Índice de tablas

Índice de figuras

1 Introducción

El TFG[2] es una actividad académica fundamental para la culminación de los estudios universitarios. Este proyecto de investigación tiene como objetivo demostrar la adquisición de conocimientos y habilidades en el área de formación de Ingeniería Informática, así como la capacidad para aplicarlos en la resolución de problemas y la realización de proyectos de investigación.

El presente TFG se enfoca en la prevención de accidentes por somnolencia en carretera, el cual ha sido seleccionado debido a su relevancia y actualidad [1]. A lo largo del trabajo se analizará y se profundizará en cómo se podría reducir el porcentaje de accidentes en carretera por somnolencia gracias a dispositivos wearable que hoy en día se han vuelto algo cotidiano, con el fin de obtener una comprensión detallada y crítica sobre el tema en cuestión.

En este trabajo se abordarán diversos aspectos teóricos y prácticos, para lo cual se llevará a cabo una investigación exhaustiva a través de la revisión bibliográfica y la realización de análisis y estudios empíricos. El resultado final será una contribución al conocimiento y una propuesta de solución a la problemática abordada creando una aplicación móvil Android conectada por Bluetooth a un Smartwatch con Wear OS.

1.1 Motivación

La somnolencia se asocia con una variedad de efectos adversos en el cuerpo que, al conducir, presenta un grave riesgo de accidente. La fatiga del conductor reduce la calidad de conducción, reduce la capacidad de respuesta y provoca distracciones. La somnolencia afecta el procesamiento de la información y la toma de decisiones, y el comportamiento se vuelve estresante, ansioso y agresivo. Un síncope es el efecto más severo de la fatiga, ya que es un breve período de somnolencia que hace que el conductor pierda el control del vehículo. Según un informe de Línea Directa [2], 8,1 millones de conductores españoles admiten privación de sueño.

El informe también encontró que quedarse dormido en un accidente al volante duplicó el riesgo de muerte, con un 3,9 % de riesgo de fatiga en comparación con un 1,9 % de riesgo de un accidente común. Este incremento está íntimamente relacionado con el riesgo de lesiones, aumentando hasta un 15,9% para la somnolencia frente al 11,2% para los accidentes comunes. Este estudio también establece un perfil donde ocurren la mayoría de los choques. Los accidentes ocurren fuera de la carretera en la madrugada del sábado al domingo de julio y suelen ser conducidos por hombres jóvenes de entre 21 y 30 años, aunque ocurre en todas las edades.

La somnolencia al volante es una de las principales causas de accidentes de tráfico en el mundo. Según la Organización Mundial de la Salud, cada año se producen más de un millón de muertes y decenas de millones de heridos. La falta de sueño, el consumo de alcohol o drogas, el estrés o los medicamentos que provocan sueño son algunos de los factores que pueden afectar a la capacidad de atención y reacción de los conductores. Para prevenir la somnolencia al volante, se recomienda dormir al menos siete horas antes de conducir, evitar las comidas pesadas, hacer paradas cada dos horas o cambiar de conductor si es posible, y no consumir sustancias que alteren el sistema nervioso. La somnolencia al

[2] Trabajo de Fin de Grado

volante no solo pone en riesgo la vida del conductor, sino también la de los demás usuarios de la vía. Por eso, es importante estar alerta y consciente al conducir, y detenerse si se siente cansancio o sueño.

Según la Dirección General de Tráfico, la somnolencia interviene, directa o indirectamente, en entre el 15 y el 30% de los accidentes de tráfico en España. Sus efectos no sólo se manifiestan por la noche, sino que también son muy numerosos los accidentes diurnos en los que la somnolencia es un factor implicado. [3]

Los smartwatches son dispositivos electrónicos que se utilizan principalmente como relojes, pero que también incluyen diversas funciones adicionales. Estos dispositivos están diseñados para ser llevados en la muñeca y suelen estar conectados a un teléfono inteligente.

Una de las funciones más populares de los smartwatches es la monitorización del sueño. Esta función permite a los usuarios hacer un seguimiento de su sueño y obtener información detallada sobre la calidad y cantidad de su descanso. Para ello, los smartwatches utilizan una serie de sensores y algoritmos que miden la actividad y los movimientos durante la noche.

Algunos de los datos que pueden ser recopilados por estos relojes durante la monitorización del sueño incluyen el tiempo total de sueño, el tiempo en cada fase del sueño (REM[3], sueño ligero y sueño profundo), la frecuencia cardíaca y la respiración. Estos datos se pueden analizar para identificar patrones de sueño irregulares y mejorar la calidad de vida en general y utilizar para generar informes detallados sobre la calidad del sueño del usuario.

La detección de la somnolencia conduciendo es un tema de investigación que ha recibido una creciente atención en los últimos años debido al alto número de accidentes de tráfico relacionados con la fatiga y la somnolencia. En este contexto, los smartwatches han surgido como una posible solución para la detección de la somnolencia conduciendo de manera no invasiva.

En la actualidad, existen diversos estudios que han abordado la detección de la somnolencia conduciendo mediante el uso de smartwatches. Estos estudios han utilizado diferentes sensores y algoritmos para medir y analizar los patrones de sueño y la actividad física durante la conducción.

Por ejemplo, algunos estudios han utilizado la aceleración y la frecuencia cardíaca para detectar la fatiga y la somnolencia durante la conducción. Otros han utilizado la variabilidad de la frecuencia cardíaca y la temperatura corporal para detectar cambios en el estado de ánimo y la alerta del conductor.

Además, se han propuesto diferentes técnicas para la detección de la somnolencia conduciendo mediante el uso de smartwatches, incluyendo el análisis de la variabilidad de la frecuencia cardíaca, la detección de movimientos oculares y la combinación de varios sensores para mejorar la precisión de la detección.

A pesar de los avances en la detección de la somnolencia conduciendo mediante el uso de smartwatches, todavía existen varios desafíos que deben ser abordados, como la precisión y la fiabilidad de los sensores, la adaptabilidad de los algoritmos a diferentes

[3] Rapid Eye Movement

condiciones de conducción y la necesidad de validar los resultados obtenidos en estudios a gran escala.

En resumen, la detección de la somnolencia conduciendo mediante el uso de smartwatches es un tema de investigación emergente que presenta un gran potencial para mejorar la seguridad vial. Sin embargo, todavía es necesario seguir investigando para mejorar la precisión y la fiabilidad de los sensores y algoritmos utilizados, así como para validar los resultados en estudios a gran escala.

1.2 Estado del arte

Estas son algunas de las aplicaciones similares al proyecto:

1.2.1 Samsung Copiloto

Se trata de una aplicación instalada en un smartwatch de Samsung que aprende de los patrones de cada conductor detectando el cansancio al volante, mientras este mantiene la atención en la carretera. Los sensores internos del reloj: giroscopio, acelerómetro y monitor de frecuencia cardíaca se encargarán de medir lo que suceda durante el trayecto. Si el reloj detecta peligro de somnolencia o se sobrepasa el máximo de horas al volante recomendadas, comenzará a emitir avisos en forma de vibración. Desgraciadamente esta aplicación se descatalogó del mercado. [4]

1.2.2 Drowsy Driving Alert

El objetivo de la aplicación *Drowsy Driving Alert* es simple, pero muy útil si deseas luchar contra el sueño mientras conduces. Cuando inicias la aplicación y presionas el botón de inicio, la aplicación detecta tu rostro y coloca un cuadrado verde alrededor.

Si detecta que tus ojos están cerrados, suena una alerta en tu teléfono, como un despertador. Después de cerrar los ojos, la aplicación envía una alerta en segundos.

Aquí no se hace uso de dispositivos externos más allá del móvil, pero su objetivo es el mismo que el de mi proyecto así que también sirve de inspiración y puede que en algún futuro se intente implementar como método adicional al de la detección de ritmo cardíaco con smartwatch. [5]

1.2.3 Nap Zapper

Una empresa llamada *Zhenjiang Welkin Electronics Co.,Ltd*. fundada en 2004, lanzó al mercado un dispositivo que va puesto en la oreja la cuál dispone de un giroscopio que al detectar que la cabeza cae hacia delante mientras se está conduciendo, suene fuertemente en la oreja y despierte evitando un peligroso accidente. [6]

1.3 Objetivos

El objetivo principal de este proyecto consiste en desarrollar una aplicación para teléfonos móviles Android, enfocada a la detección de somnolencia y/o fatiga mediante el uso de dispositivos externos *wearables* con Wear OS.

Se ha decidido utilizar un *smartwatch* ya que es un dispositivo que permite hacer mediciones y análisis de variables fisiológicas del individuo que lo lleva puesto. Estas variables pueden ser la frecuencia cardíaca, nivel de saturación de oxígeno en sangre, calidad de sueño, nivel de estrés, etc.

Los objetivos de este trabajo se detallan en:

- Análisis de distintos proyectos parecidos a este.
- Creación de una aplicación para *Samsung Galaxy Watch 5* capaz de leer los latidos gracias a sus sensores para enviarlos a la APP móvil.
- Creación de una aplicación desarrollada en *Kotlin* capaz de detectar un Smartwatch mediante la tecnología BLE[4].
- Implementar un algoritmo de detección de somnolencia que, usando las lecturas previas del reloj, pueda detectar el estado del usuario y avisarlo con una vibración en el reloj o un sonido en su móvil.
- Visualización de los latidos obtenidos en pantalla mediante un gráfico de líneas en tiempo real.

1.4 Organización de la memoria

En este apartado se pretende mostrar cuáles son las partes en las que se ha dividido la memoria con tal de explicar el proceso de desarrollo del proyecto:

1) Introducción: En esta sección se hace una descripción general del proyecto, planteando los problemas y exponiendo los objetivos principales.
2) Tecnologías utilizadas: Aquí se detallará tanto el software como el hardware empleado.
3) Diseño: En esta sección se explicará la estructura del proyecto utilizada, las decisiones de diseño, funcionalidades, etc.
4) Implementación y algoritmo: En este capítulo se expondrán las ideas para implementar toda la aplicación, tanto para poder detectar los latidos como el método para detectar la somnolencia.
5) Conclusiones: Por último, se hablará de cómo ha ido el proyecto, dificultades, metas y posibles ideas para futuras mejoras de Driving Angel.

[4] Bluetooth Low Energy

2 Tecnologías utilizadas

En esta sección se realiza un estudio del hardware y software empleados en el desarrollo del trabajo.

2.1 Lenguaje: Kotlin

Crear una aplicación para smartwatches conectada al móvil utilizando Android como plataforma de desarrollo en lugar de iOS[5] puede tener algunas ventajas:

- Mayor alcance de mercado: Android tiene una mayor cuota de mercado en comparación con iOS en términos de dispositivos móviles, incluyendo smartphones y smartwatches. Por lo tanto, desarrollar una aplicación para smartwatches en Android puede permitirte llegar a un público más amplio y potencialmente aumentar la base de usuarios de tu aplicación.
- Mayor flexibilidad de hardware: Android permite una mayor diversidad de fabricantes y modelos de smartwatches en comparación con iOS, lo que brinda a los desarrolladores más opciones en términos de hardware.
- Mayor acceso a APIs[6] y servicios de Google: Como plataforma de Google, Android ofrece un mayor acceso a las APIs y servicios de Google, lo que puede permitirte integrar funcionalidades avanzadas como las que se han utilizado para acceder y medir los latidos.

Como los dispositivos iOS suelen ser más caros y con un ecosistema más cerrado y teniendo en cuenta que en la carrera no se hace nada con Swift se decidió decantar esta opción.

Kotlin es un lenguaje de programación moderno que ha ganado popularidad en el desarrollo de aplicaciones para la plataforma Android, ya que desde hace relativamente poco tiempo Google estableció Kotlin como el lenguaje oficial para dispositivos Android [7]. Algunas de las ventajas de por qué se utilizó Kotlin en lugar de Java para programar aplicaciones son:

- Interoperabilidad con Java: Puedes llamar a código Java desde Kotlin y viceversa
- Sintaxis concisa y expresiva: Reduce la cantidad de código necesario para lograr una funcionalidad similar a Java
- Nulabilidad de tipo: Kotlin aborda el problema de referencias nulas (NullPointerExceptions) que es común en Java, mediante su sistema de tipos que distingue entre referencias nulas y no nulas de forma estática.
- Programación funcional: Kotlin admite características de programación funcional, como funciones de orden superior, lambdas y flujo de datos inmutables, lo que permite escribir código más conciso

[5] iPhone Operative System

[6] Application Program Interface

- Soporte para características modernas de Android: Kotlin se ha diseñado específicamente para el desarrollo de aplicaciones Android y ofrece un excelente soporte para las últimas características y APIs de Android.

2.2 IDE[7]: Android Studio

Android Studio es el IDE oficial para la plataforma Android, basado en IntelliJ IDEA, otro IDE muy popular. Esto significa que está optimizado para el desarrollo de aplicaciones en la plataforma Android y se mantiene actualizado con las últimas características y mejoras de Android [8].

En este aspecto no hubo mucha complicación para elegirlo, ya que Android Studio facilita diversas cosas:

- Integración con herramientas de desarrollo de Android: Android Studio ofrece una amplia gama de herramientas específicas para el desarrollo de aplicaciones Android, como emuladores de dispositivos Android, herramientas de diseño de interfaces de usuario (UI), analizadores de rendimiento, herramientas de depuración avanzada, integración con Google Play Services, y muchas más.
- Integración con herramientas de desarrollo de Android: Android Studio ofrece una amplia gama de herramientas específicas para el desarrollo de aplicaciones Android, como emuladores de dispositivos Android, herramientas de diseño de interfaces de usuario (UI), analizadores de rendimiento, herramientas de depuración avanzada, integración con Google Play Services, y muchas más.
- Poder crear y organizar elementos gráficos para la interfaz de la APP sin utilizar código, además de realizar renderizados de layouts en tiempo real.
- Integración de control de versiones como GitHub.
- Estas herramientas facilitan el desarrollo y la depuración de aplicaciones Android y emuladores tanto móviles como Smartwatches, cosa que fue clave para el proyecto ya que desde el departamento se compró un reloj el cuál tardó bastante en llegar.

Aquí se muestra una captura de pantalla de la interfaz de Android Studio:

[7] Integrated Development Environment

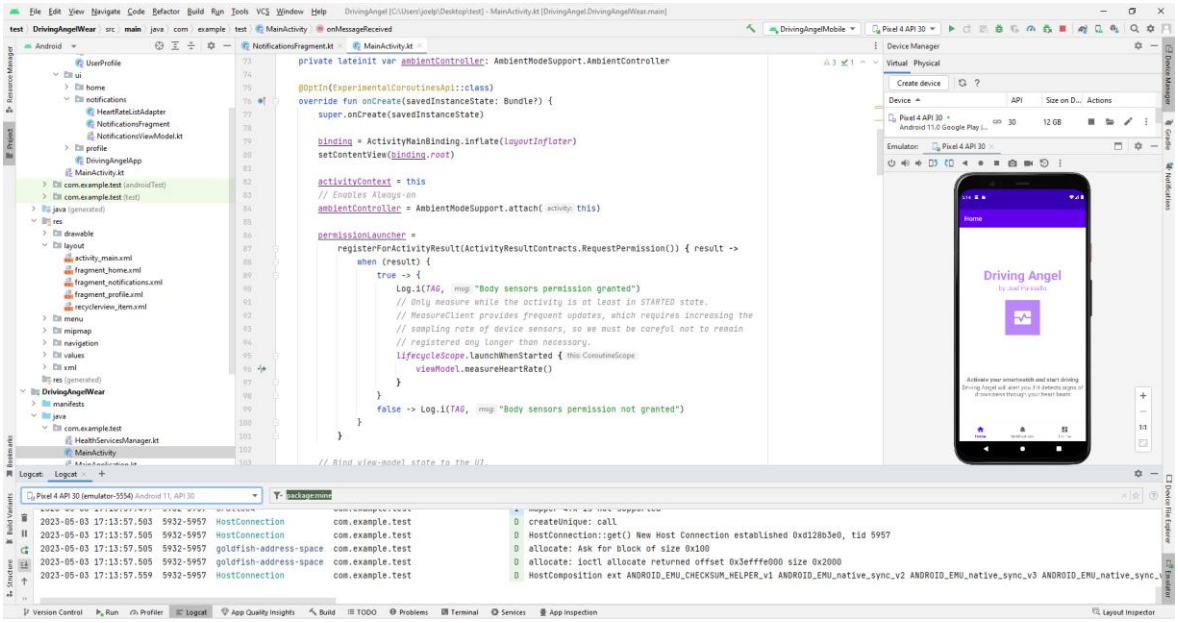

Figura 1. Interfaz principal Android Studio

- A la izquierda, la estructura del proyecto con los paquetes, clases, layouts, etc.
- En el centro de la pantalla está el código para editar con las distintas pestañas de archivos
- A la derecha está el Gestor de dispositivos, para controlar y editar configuraciones tanto de los emuladores como dispositivos físicos. Justo debajo está la vista en ejecución de la aplicación, pudiéndose poner a pantalla completa
- En la parte inferior hay varias pestañas, como el control de versiones, información de la construcción y ejecución del programa y logs para poder depurar el código

2.3 Base de Datos: SQLite

Room es un ORM[8] creada por el equipo de Android en Google, que simplifica la tarea de trabajar con bases de datos en Android [9].

Actúa de intermediaria con la APP. Se tendrá una base de datos que le devolverá a la APP los DAO[9], que son los encargados de persistir la información en la base de datos y de devolver las **entities**, que serán las encargadas de devolver la información que se ha ido almacenando. Room usa el DAO para enviar consultas a la base de datos SQLite[10].

[8] Object-Relational mapping

[9] Data Access Objects

[10] Structured Query Language

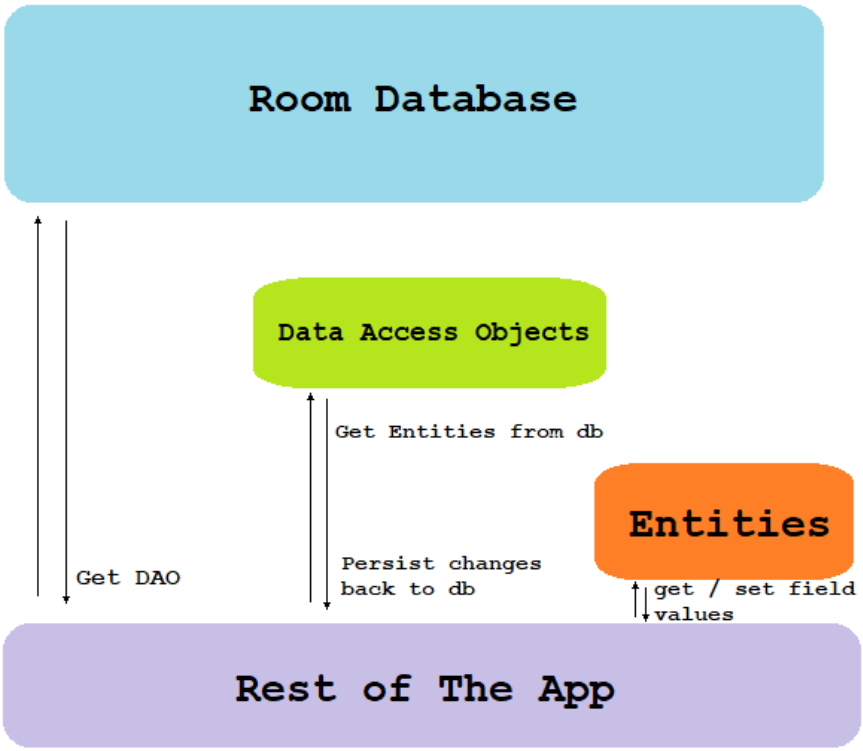

Figura 2. Diagrama de la arquitectura de la biblioteca de Room

Aunque en principio no se pretende persistir los datos ya que cada vez que se encienda la APP empezará a hacer los cálculos como nuevos. Se ha decidido utilizar una base de datos para realizar estas operaciones de manera eficiente y cumplir con los principios ACID[11]. Quizá en un futuro se utilice una pestaña para ver un resumen gráfico de las veces que se ha realizado un trayecto con nuestra aplicación y para entonces esta base de datos ya estará implementada.

2.4 Conectividad: Data Layer

Para que las dos aplicaciones se puedan comunicar se decidió utilizar la API[12] de Data Layer [10], que usa la clase Wearable para obtener instancias de la clase MessageClient. Esta API solo puede enviar mensajes y sincronizar datos con dispositivos Android o relojes Wear OS que estén vinculados entre sí.

No se produce ninguna sincronización entre las APPs móviles y para wearables. Los mensajes son un mecanismo de comunicación unidireccional útil para las llamadas de procedimientos remotos (RPC).

Los distintos dispositivos conectados son conocidos como Nodos. Para detectar nodos capaces a medida que se conectan al dispositivo wearable, se registra un OnCapabilityChangedListener para la escucha de estos. Después de identificar el nodo, se

[11] Atomicidad, Consistencia, Aislamiento y Durabilidad

[12] Application Program Interface

verifica que esté disponible y se determina adónde enviar el mensaje. Esta llamada es síncrona y bloquea el procesamiento hasta que el sistema encola el mensaje para su entrega.

Como todos los usuarios con dispositivos Wear OS tienen su aplicación por defecto, dejamos que empareje dicho dispositivo mediante esa APP y así evitar repetir funciones.

2.5 Smartwatch: Samsung Galaxy Watch 5

A la hora de elegir un Smartwatch para crear la aplicación había que buscar uno que fuera compatible con el lenguaje utilizado. Teniendo en cuenta que existen varios sistemas operativos para relojes inteligentes en el mercado:

- TizenOS: es un sistema operativo de código abierto basado en Linux que se utiliza en dispositivos móviles y wearables de Samsung antiguos.
- WatchOS: es un sistema operativo exclusivo para los relojes inteligentes de Apple. Es muy fácil de usar y tiene una gran cantidad de aplicaciones disponibles.
- Wear OS: es una plataforma diseñada específicamente para dispositivos wearables que se centra en Android creado por Google y ofrece un aspecto y un conjunto de funcionalidades únicas.

Sabiendo esto, se decantó por los relojes que tuvieran el SO de Wear OS con lo cual se compró el Samsung Galaxy Watch 5, que actualmente es la última creación en relojes de Samsung. Aquí están sus características principales:

Figura 3. Samsung Galaxy Watch 5

Sistema Operativo	WearOS 4.5 con OneUI Watch
Procesador	Exynos W920
RAM[13]	1,5 GB
Almacenamiento	16 GB
Batería	284 mAh
Pantalla	Super AMOLED de 1,2 pulgadas
Sensores	NFC[14], LTE[15], Bluetooth, acelerómetro, barómetro, sensor de luz, giroscopio, sensor óptico de latidos, sensor eléctrico de latidos y sensor de análisis de impedancia bioeléctrica

Tabla 1. Especificaciones del Samsung Galaxy Watch 5

[13] Random Access Memory

[14] Near Field Communication

[15] Long Term Evolution

3 Diseño

La arquitectura de este trabajo se divide en 2 grandes bloques que se comunican entre ellos por medio de BLE, la aplicación móvil y la del reloj.

3.1 Driving Angel Mobile

Para esta parte se ha seguido la arquitectura recomendada para aplicaciones móviles por *Android Developers*.

Básicamente hay una capa de UI[16] que muestra los datos de la aplicación en la pantalla y la capa de datos que contiene la lógica de la aplicación y expone los datos.

Este es un pequeño diagrama sobre los componentes usados y su funcionamiento en conjunto:

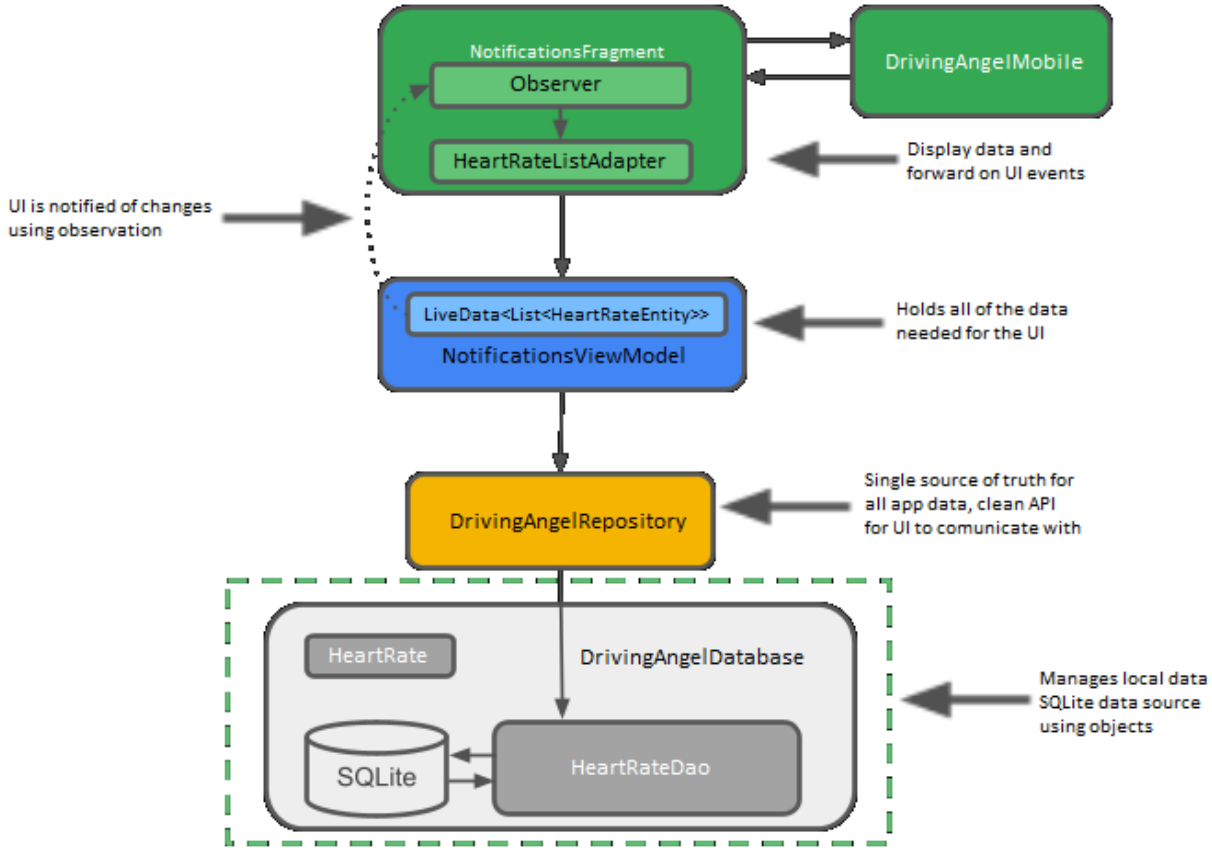

Figura 4. Diagrama de componentes

LiveData es una clase en Android que proporciona un contenedor de datos observable. Su objetivo principal es mantener y almacenar en caché la última versión de los datos y notificar automáticamente a los observadores cuando los datos cambian.

[16] User Interface

El ViewModel en Android actúa como un intermediario entre la UI y el repositorio de datos. Su función principal es proporcionar los datos necesarios para que la interfaz de usuario funcione correctamente.

El repositorio es una clase que administra múltiples fuentes de datos.

La entidad es una clase anotada que describe una tabla de base de datos cuando se trabaja con Room.

La base de datos de Room simplifica el trabajo con la base de datos porque sirve como punto de acceso a la base de datos SQLite subyacente. Utiliza el DAO para enviar consultas a la base de datos.

La base de datos SQLite es el almacenamiento del dispositivo. Room crea y mantiene esta base de datos por ti.

DAO es el objeto de acceso a datos que mapea búsquedas de SQL a funciones. Al utilizar DAO, simplemente se llaman los métodos y Room se encarga del resto.

La capa de la UI consta de:

- Un menú de botones de navegación para cambiar entre pestañas dentro de la APP
- Contenedores de estados ViewModel que retienen los datos, los exponen a la UI y controlan la lógica

La pestaña Home (**Figura 5**) actúa simplemente como portada de la APP, para dar información básica de su funcionamiento.

La pestaña de Notificaciones consta de una lista con los latidos que se han registrado y una gráfica de líneas que se actualizan en tiempo real gracias al componente LiveData, ya que siempre conserva en caché la última versión de los datos, y notifica a los observadores cuando estos han cambiado.

Debajo está el botón para comprobar si están conectados los dos dispositivos para entonces comenzar con la medición.

La pestaña del Perfil se creó cuando se pensaba en la primera aproximación del algoritmo de detección de somnolencia, sin embargo, al final no se han utilizado tales parámetros como el sexo, la edad o saber si una persona es atlética o no. No obstante, se ha decidido dejar esa pestaña por si en un futuro la aplicación evoluciona y se pueda hacer uso de dichos parámetros adicionales para poder dar una aproximación más precisa y personalizada del posible sueño al volante de los usuarios de Driving Angel.

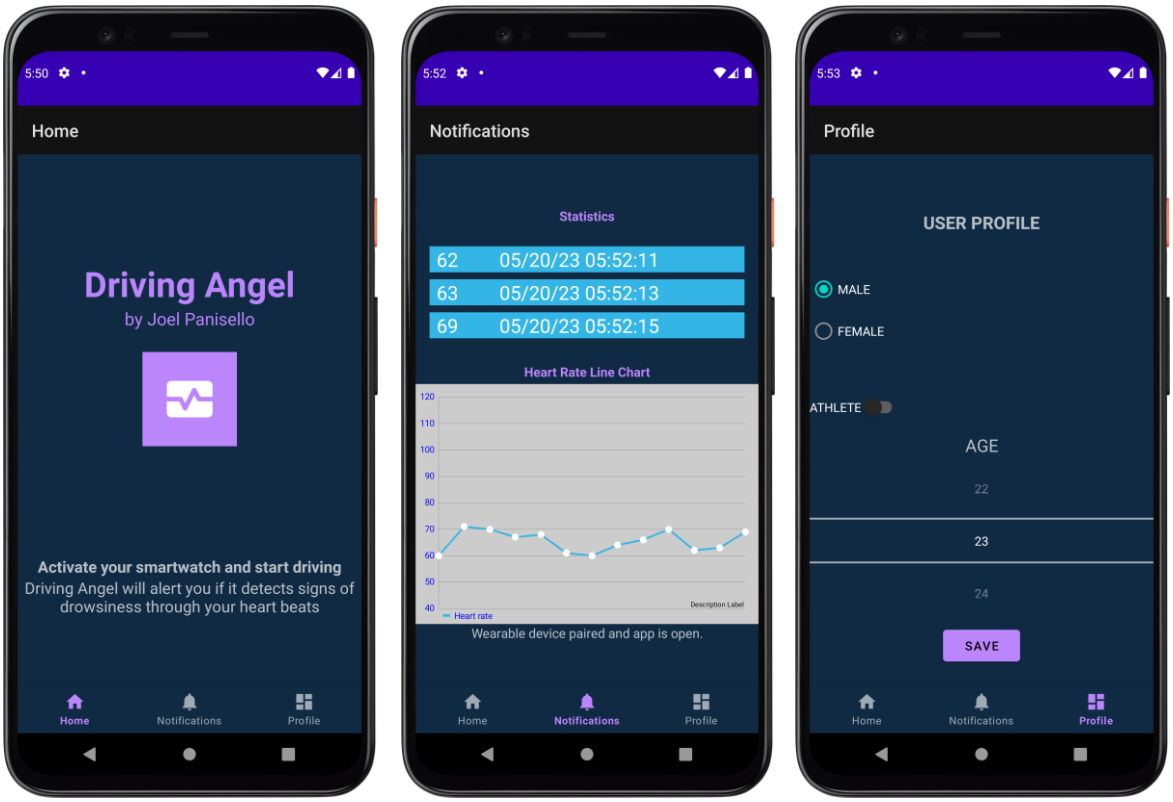

Figura 5. Vista principal móvil

El flujo de trabajo de la aplicación móvil sería el siguiente:

1. Pulsar el botón “Check for connected wearables” para iniciar el emparejamiento entre móvil y reloj enviando un mensaje de comprobación esperando respuesta del reloj.
2. Una vez se ha recibido la correcta respuesta, el móvil comenzará a recibir los latidos conforme el reloj los tenga disponibles.
3. Al recibirlos, se almacenará la fecha y el valor de pulsaciones por minuto en la base de datos.
4. Al añadirse a la base de datos, automáticamente gracias al ViewModel se actualizará la lista y se añadirá el punto en la gráfica de líneas.
5. A los 15 minutos de recibir latidos se habrá guardado la media, desviación y coeficiente de variación dinámicamente como estado inicial.
6. Después de este tiempo, de todos los datos que se van leyendo, se irá calculando lo mismo cada 1 minuto comparándolo con el estado inicial. Si las medias difieren en un 6%, el coeficiente de variación en un 10% o más y la desviación estándar no pasa de los 10 LPM, se procederá a alertar al usuario de forma sonora desde el teléfono móvil para que reaccione. Más adelante se justifican estas condiciones.

La función más relevante en esta aplicación es la de detección y alerta del sueño, se procede a explicarla más detalladamente:

```
private fun sleepDetection(heartRates: List<HeartRateEntity>) {
    CoroutineScope(Dispatchers.IO).launch {
        runBlocking {
            if (initialState) {     // First period
                if (Date().before(finalDate)) {
                    addNumber(heartRates.last().heart_rate)
                    currentVariationCoefficient =
variationCoefficientCalculation()
                } else {
                    val toneGenerator =
ToneGenerator(AudioManager.STREAM_MUSIC,
                        ToneGenerator.MAX_VOLUME)
                    toneGenerator.startTone(
                        ToneGenerator.TONE_CDMA_SOFT_ERROR_LITE,
                        3000
                    )
                    (activityContext as Activity).runOnUiThread {
                        Toast.makeText(
                            activityContext,
                            "initialMean =
${BigDecimal(currentMean).setScale(2,

RoundingMode.HALF_EVEN).toDouble()}",
                            Toast.LENGTH_LONG
                        ).show()
                    }
                    initialState = false
                    initialMean = currentMean
                    initialStandardDeviation =
currentStandardDeviation
                    initialVariationCoefficient =
currentVariationCoefficient
                    avg = 0.0
                    sum = 0.0
                    squareSum = 0.0
                    quantity = 0
                    finalDate = addXMin(Date(), 1)
                }

            } else {    //compare initial and current states
                if (Date().before(restDate)){
                    if (Date().before(finalDate)) {
                        addNumber(heartRates.last().heart_rate)
                        currentVariationCoefficient =
variationCoefficientCalculation()
                        if (meanDifference(initialMean,
currentMean) < -6 &&

calculateDecrease(initialVariationCoefficient,
                            currentVariationCoefficient) <= -10.0 &&
                            currentStandardDeviation < 10) {
                            val toneGenerator =
ToneGenerator(AudioManager.STREAM_MUSIC, ToneGenerator.MAX_VOLUME)
                            toneGenerator.startTone(
ToneGenerator.TONE_CDMA_CALL_SIGNAL_ISDN_PING_RING, 3000)
```

```
(activityContext as Activity).runOnUiThread {
                                    Toast.makeText(
                                        activityContext,
                                        "curSD =
${BigDecimal(currentStandardDeviation)
    .setScale(2, RoundingMode.HALF_EVEN).toDouble()} " +
                                        "curCV =
${BigDecimal(currentVariationCoefficient)
    .setScale(2, RoundingMode.HALF_EVEN).toDouble()} " +
                                        "curMN =
${BigDecimal(currentMean)
    .setScale(2, RoundingMode.HALF_EVEN).toDouble()}",
                                        Toast.LENGTH_SHORT
                                    ).show()
                                }
                            }
                        } else {
                            avg = 0.0
                            sum = 0.0
                            squareSum = 0.0
                            quantity = 0
                            finalDate = addXMin(Date(), 2)
                        }
                    } else {
                        val toneGenerator =
ToneGenerator(AudioManager.STREAM_MUSIC,
                            ToneGenerator.MAX_VOLUME)
                        toneGenerator.startTone(
                            ToneGenerator.TONE_CDMA_ALERT_CALL_GUARD,
                            3000
                        )
                        (activityContext as Activity).runOnUiThread {
                            Toast.makeText(
                                activityContext,
                                "Stop at the nearest service area",
                                Toast.LENGTH_SHORT
                            ).show()
                        }
                    }
                }
            }
            try {
                delay(25)
            } catch (e: InterruptedException) {
                e.printStackTrace()
            }
        }
    }
```

Código 1. Función detección del sueño

Esta función se encarga de realizar la detección de somnolencia en base a ciertas características. Se le pasa por parámetro una lista de frecuencias cardíacas. A continuación, se explica paso a paso lo que ocurre en la función:

1. Se crea un nuevo *CoroutineScope* en el hilo IO para poder realizar operaciones asíncronas.

2. Se ejecuta un bloque *runBlocking*, que espera a que se completen todas las operaciones dentro de él antes de continuar.
3. Se comprueba si la variable *initialState* es verdadera, lo que significa que se encuentra en el primer período de medición.
4. Si es el primer período de medición, se comprueba si la fecha actual es anterior a la fecha final, y si es así, se agrega la última frecuencia cardíaca de la lista de frecuencias cardíacas recibidas y se calcula el coeficiente de variación, media y desviación estándar actuales.
5. Si la fecha actual es posterior a la fecha final, se crea un tono de sonido y se muestra un mensaje *Toast* que muestra la media inicial, y se actualizan las variables de estado. Además, se reinician algunas variables necesarias para realizar nuevos cálculos y se establece una nueva fecha final para el siguiente período de medición.
6. Si no es el primer período de medición, se comprueba si la fecha actual es anterior a la fecha de descanso.
7. Si la fecha actual es anterior a la fecha final, se agrega la última frecuencia cardíaca de la lista de frecuencias cardíacas recibidas y se calcula el coeficiente de variación actual. Además, se comprueba si la diferencia de medias entre la media inicial y la media actual es menor que -6, si la disminución del coeficiente de variación es de al menos de un 10%, y si la desviación estándar actual es menor que 10. Si se cumplen estas condiciones, se crea un tono de sonido y se muestra un mensaje *Toast* con información sobre las variables actuales para alertar al conductor de un posible accidente por somnolencia.
8. Si la fecha actual es posterior a la fecha final, se reinician algunas variables necesarias para realizar nuevos cálculos y se establece una nueva fecha final para el siguiente período de medición.
9. Si la fecha actual es posterior a la fecha de descanso, se crea un tono de sonido y se muestra un mensaje *Toast* con la información de "Parar en la zona de servicio más cercana".
10. Se utiliza una pausa de 25 milisegundos para evitar una sobrecarga de operaciones.

Hay que comentar también que se ha desarrollado la aplicación móvil ofreciendo dos estilos diferentes para adaptarse al modo oscuro y claro del teléfono. El modo oscuro y claro son dos opciones de visualización que permiten a los usuarios personalizar la apariencia de la aplicación según sus preferencias.

Cuando se activa el modo oscuro en el dispositivo, la aplicación cambia automáticamente a un tema oscuro que utiliza un fondo oscuro con texto y elementos de navegación en tonos claros. Esto ayuda a reducir la fatiga visual en condiciones de poca luz y mejora la legibilidad del contenido.

Por otro lado, cuando se activa el modo claro en el dispositivo, la aplicación cambia a un tema claro que utiliza un fondo claro con texto y elementos de navegación en tonos más oscuros. Esto ayuda a mejorar la legibilidad en condiciones de mucha luz y a crear una experiencia visual más agradable.

Ambos temas han sido diseñados para proporcionar una experiencia de usuario consistente y atractiva. La aplicación utiliza técnicas de diseño para asegurarse de que la interfaz de usuario sea clara y fácil de navegar.

Ya que los especialistas afirman que la exposición a la luz suprime la secreción de melatonina, una hormona que influye en el ciclo natural del sueño [11]. Se optó por poner el fondo en tonos azules para potenciar el efecto por si en algún momento se decide conducir de noche con la aplicación puesta. Ayudaría a mantener-se despierto, que es el objetivo del trabajo.

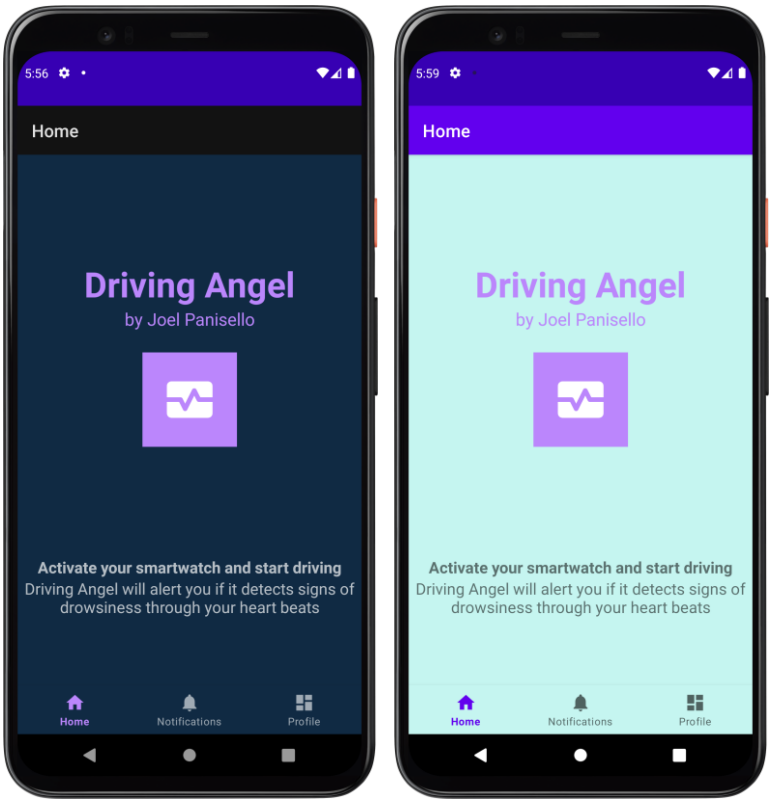

Figura 6. Diferencia Vista en Modo Claro y Oscuro

Para darle una vuelta más a estos temas visuales, se ha creado una nueva funcionalidad para la aplicación:

La nueva funcionalidad implementada en esta aplicación móvil permite que el sistema utilice el sensor de luminosidad incorporado en el teléfono para ajustar automáticamente el modo de visualización entre claro y oscuro. El sensor de luminosidad captura la cantidad de luz ambiental que rodea al dispositivo y la aplicación interpreta estos datos para determinar el nivel de brillo óptimo para la pantalla.

Cuando el sensor de luminosidad detecta una alta luminosidad ambiental, la aplicación cambia al modo claro, que presenta un fondo de pantalla y elementos de interfaz de usuario en tonos más claros. Esto ayuda a mejorar la legibilidad y visibilidad de los elementos en situaciones de mucha luz, como estar al aire libre en un día soleado.

Por otro lado, cuando el sensor de luminosidad detecta una baja luminosidad ambiental, la aplicación cambia automáticamente al modo oscuro, que presenta un fondo

de pantalla y elementos de interfaz de usuario en tonos más oscuros. Esto ayuda a reducir la intensidad de la luz emitida por la pantalla, disminuyendo la fatiga visual en entornos con poca luz o en situaciones nocturnas.

La inclusión de esta funcionalidad en la aplicación móvil ofrece varias ventajas y beneficios para los usuarios:

1. Reducción de la fatiga visual: Al adaptarse automáticamente a las condiciones de iluminación ambiental, la aplicación ayuda a reducir la fatiga visual causada por la exposición a pantallas brillantes durante períodos prolongados.
2. Mayor comodidad visual: Al ajustar el modo de visualización según las condiciones de luz, la aplicación proporciona una experiencia visual más cómoda y agradable para los usuarios, evitando la necesidad de realizar cambios manuales.
3. Mejora de la legibilidad: La capacidad de cambiar entre modos claro y oscuro en función de la luminosidad ambiental mejora la legibilidad y visibilidad de los elementos de la interfaz de usuario, lo que facilita la lectura y la interacción con la aplicación en diferentes entornos.
4. Ahorro de energía: El cambio al modo oscuro en condiciones de poca luz puede contribuir a un menor consumo de energía en dispositivos con pantallas OLED, ya que los píxeles oscuros requieren menos energía que los píxeles brillantes.

3.2 Driving Angel Wear

Siguiendo los principios básicos de proyectos oficiales de Android [12] se ha creado una aplicación la cual permite hacer lecturas de las pulsaciones del corazón mediante el uso de una librería de código abierto oficial de Android que aún está en fase beta [13].

Gracias a esto podemos iniciar las lecturas del reloj y estar observando hasta que se reciban los datos en un formato entendible.

A continuación, se muestra la vista principal de la aplicación:

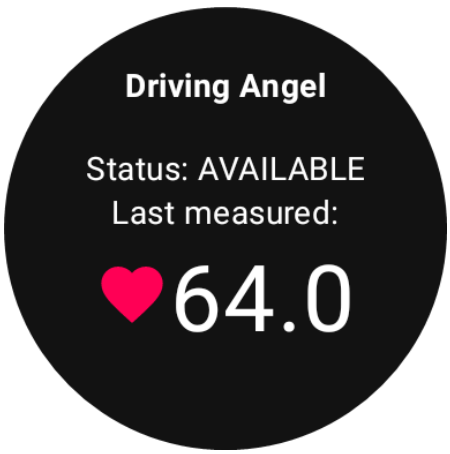

Figura 7. Vista principal smartwatch

Como se puede ver en la **Figura 7,** hay 3 cosas a tener en cuenta:

1. Una vez se inicia la aplicación ya comienza a medir las pulsaciones siempre que el reloj permita este tipo de mediciones. En caso de que no se puedan hacer estas mediciones, como por ejemplo que no esté el reloj puesto en la muñeca del usuario aparecerá el estatus como "UNAVAILABLE". Se puede ver en la **Figura 3**.
2. También se puede verificar que el estado de dichas mediciones siga disponible.
3. Hay un texto el cual indica el estado de la conexión entre el smartwatch y el móvil.

Si se han emparejado los dispositivos correctamente, una vez se realice una lectura de los latidos, el reloj enviará un mensaje con los datos y cuando el móvil los reciba los guardará y mostrará en una gráfica.

El mecanismo de comunicación es el mismo que en la otra aplicación, una vez se recibe desde el teléfono el mensaje inicial para comprobar si los dos dispositivos tienen la aplicación abierta, el reloj comenzará a enviar los latidos siempre que estén disponibles.

Aquí se muestra el código de envío de latidos al móvil:

```
private fun sendHR(heartRate: String) {
    if (mobileDeviceConnected) {
        val nodeId: String = messageEvent?.sourceNodeId!!
        val payload: ByteArray = heartRate.toByteArray()

        val sendMessageTask =
            Wearable.getMessageClient(activityContext!!)
                .sendMessage(nodeId, MESSAGE_ITEM_RECEIVED_PATH,
payload)

        binding.connectionStatusText.visibility = View.GONE

        sendMessageTask.addOnCompleteListener {
            if (it.isSuccessful) {
                Log.d("send1", "Message sent successfully")
                val sbTemp = StringBuilder()
                sbTemp.append("\n")
                sbTemp.append(heartRate)
                sbTemp.append(" (Sent to mobile)")
                Log.d("receive1", " $sbTemp")
            } else {
                Log.d("send1", "Message failed.")
            }
        }
    }
}
```

Código 2. Función de envío de frecuencia cardíaca

Aquí está el flujo de la función:

1. Verifica si el dispositivo móvil está conectado mediante la variable booleana *mobileDeviceConnected*.

2. Obtiene el ID del nodo (dispositivo) fuente del evento de mensaje a través de la propiedad *messageEvent?.sourceNodeId!!*, donde *messageEvent* es la variable la cual recibía el nodo al hacer el handshake.
3. Convierte la frecuencia cardíaca heartRate a una tabla de bytes usando la función *toByteArray()*.
4. Envía el mensaje usando el cliente de mensajes de Wearable API mediante la función *sendMessage()* que toma como parámetros el ID del nodo destino, la ruta del mensaje y el payload (los bytes de la frecuencia cardíaca).
5. Oculta el texto de conexión en la interfaz de usuario mediante la modificación de la visibilidad de *binding.connectionStatusText*.
6. Añade un listener *addOnCompleteListener* al envío del mensaje para verificar si el envío fue exitoso o no.
    a. Si el envío es exitoso, muestra un mensaje de registro en la consola con la etiqueta "send1" indicando que el mensaje fue enviado correctamente, y agrega la frecuencia cardíaca enviada a un *StringBuilder* con un formato específico.
    b. Si el envío falla, muestra un mensaje de registro en la consola con la etiqueta "send1" indicando que el mensaje ha fallado.

Para el acceso y obtención de latidos se usa *HealthServicesClient*, este es el código más importante:

```
fun heartRateMeasureFlow() = callbackFlow {
    val callback = object : MeasureCallback {
        override fun onAvailabilityChanged(dataType: DeltaDataType<*,
*>, availability: Availability) {
                        if (availability is DataTypeAvailability) {

trySendBlocking(MeasureMessage.MeasureAvailability(availability))
            }
        }
        override fun onDataReceived(data: DataPointContainer) {
            val heartRateBpm = data.getData(DataType.HEART_RATE_BPM)
            trySendBlocking(MeasureMessage.MeasureData(heartRateBpm))
        }
    }
    Log.d(TAG, "Registering for data")
    measureClient.registerMeasureCallback(DataType.HEART_RATE_BPM,
callback)

    awaitClose {
        Log.d(TAG, "Unregistering for data")
        runBlocking {

measureClient.unregisterMeasureCallbackAsync(DataType.HEART_RATE_BPM,
callback)
        }
    }
}
```

Código 3. Función de obtención de latidos por minuto

Esta función en Kotlin define un cold flow utilizando la función *callbackFlow* de la biblioteca de Kotlin Coroutines. Este flujo se utiliza para registrar un callback que recibe datos de frecuencia cardíaca y luego emite mensajes en función de esos datos. Cuando el flujo se activa, se registra un callback para recibir datos de frecuencia cardíaca utilizando la clase *MeasureCallback*, que implementa dos métodos: *onAvailabilityChanged* y *onDataReceived.*

El método *onAvailabilityChanged* se invoca cuando hay un cambio en la disponibilidad de los datos de frecuencia cardíaca, y si la disponibilidad es del tipo *DataTypeAvailability*, se envía un mensaje a través del flujo utilizando la función *trySendBlocking*. El mensaje enviado es de tipo *MeasureMessage.MeasureAvailability* y contiene el objeto *availability* como información.

El método *onDataReceived* se invoca cuando se reciben datos de frecuencia cardíaca, y se obtiene el valor de la frecuencia cardíaca en bpm (beats per minute) a partir de los datos utilizando *data.getData*(DataType.HEART_RATE_BPM). Luego, se envía un mensaje a través del flujo utilizando la función *trySendBlocking*. El mensaje enviado es de tipo *MeasureMessage.MeasureData* y contiene el valor de la frecuencia cardíaca como información.

4 Implementación y Algoritmo

Hoy en día muchas personas disponen de relojes inteligentes que miden las pulsaciones por minuto. Se decidió trabajar con dispositivos compatibles con Wear OS por la facilidad que dan para acceder a dichos sensores a la hora de programar. Se ha decidido trabajar solamente con valores de latidos por minuto por varios motivos:

1- Cada vez es más frecuente ver a personas con este tipo de relojes, pero la gran mayoría no disponen de un sensor de electrocardiogramas, simplemente se basan en una técnica llamada fotopletismografía [14]. Esta técnica se basa en un principio sencillo basado luces LED verdes combinados con fotodiodos sensibles a la luz para detectar la cantidad de sangre que fluye a través de su muñeca en un momento dado. Cuando el corazón late, el flujo arterial en la muñeca y la absorción de la luz verde es mayor. Entre latidos, es menos. Mediante el parpadeo de sus luces LED cientos de veces por segundo, el reloj es capaz de calcular el número de veces que el corazón late cada minuto o lo que es lo mismo la frecuencia cardíaca.

   Con lo cual, aunque podría haber sido más preciso, pero más lento también, se ha descartado la lectura de electrocardiogramas para que la aplicación esté disponible para más personas.

2- La otra razón, aunque menos importante, ya que la idea era que fuera accesible a más gente, ha sido la imposibilidad de acceder a estos sensores del Samsung Galaxy Watch 5. Era necesario pedir una clave de acceso a Samsung para poder acceder a su librería donde ya tienen algoritmos preparados para la obtención de los datos crudos del electrocardiograma y por desgracia solo dan acceso a empresas relacionadas con el sector de la salud y el deporte. Tampoco hay una librería que generalice el acceso a un sensor de electrocardiogramas y por tanto la aplicación se hubiera basado solamente en los relojes de Samsung y no era la intención.

Una vez aclarado que se pretende hacer una detección del sueño al volante utilizando solamente sensores de fotopletismografía, vamos a hablar de qué factores pueden ser determinantes.

4.1 Aproximación del algoritmo

En un inicio se había contemplado la simple idea de buscar estudios que relacionaran el sexo, edad y condición física con las pulsaciones normales que debería tener cada individuo en un estado de reposo. Por ese motivo la aplicación contiene una pestaña de perfil la cual permite especificar estos parámetros y crear un algoritmo para poder predecir según estos valores a que pulsaciones podría llegar a bajar si la persona se adormeciera. Si que es cierto que hay algunos estudios que indican un abanico de valores para cada grupo, pero aun así es impreciso decidir los valores normales de este modo.

Después de un tiempo dándole vueltas al asunto se descartó la idea porque hay muchos factores que determinan cuál es el estado correcto de pulsaciones en cada persona y sería poco efectivo generalizar por grupos.

4.2 Algoritmo final

Si se quiere llegar a tantas personas con condiciones distintas se tenía que llegar a una solución para poder establecer un rango de pulsaciones normal para cada individuo de forma personalizada. Esta forma fue establecer un periodo inicial de 15 a 30 minutos [15] con el que sabremos que dicho individuo acaba de conectar la aplicación y va a comenzar su trayecto. De esta forma se irá midiendo los latidos y se calculará de forma acumulada la

media, desviación estándar y coeficiente de variación durante todo ese periodo. Cuando termine dicho periodo se empezará a calcular otra vez lo mismo, pero se irá comparando con el estado inicial.

No hay un valor específico de coeficiente de variación en latidos por minuto que se pueda utilizar para comparar el estado despierto de un conductor con su estado actual. La HRV[17] y el CV[18] pueden variar según la edad, el género, la condición física y otros factores individuales.

Sin embargo, se ha encontrado que los conductores somnolientos tienen una disminución significativa en la HRV y en el CV de los LPM[19] en comparación con los conductores despiertos. En este proyecto, después de consultar varios artículos científicos se ha considerado que una disminución del CV de los LPM del 10% o más en comparación con el estado despierto del conductor puede indicar una disminución de la HRV y un mayor riesgo de somnolencia al conducir.

En resumen, las condiciones en conjunto para detectar somnolencia son las siguientes:

1. Una disminución de un 6% entre las medias, que es un porcentaje ligeramente inferior al mencionado en un artículo científico [16].
2. Una disminución del CV de los LPM del 10% o más en comparación con el estado despierto
3. Una desviación estándar de 10 LPM

Es importante tener en cuenta que el análisis de la HRV y del CV de los LPM debe ser realizado por un profesional médico capacitado en la interpretación de estos datos, y que estos valores deben ser evaluados en conjunto con otros indicadores clínicos y la historia médica del usuario. Además, el monitoreo de la HRV y del CV de los LPM no debe ser utilizado de manera aislada para tomar decisiones clínicas, sino que debe ser utilizado como parte de una evaluación completa de la somnolencia al conducir.

[17] Heart Rate Variability

[18] Coeficiente de Variación

[19] Latidos por Minuto

5 Evaluación

La evaluación del Trabajo de Fin de Grado se llevó a cabo mediante una serie de casos de prueba diseñados para verificar el funcionamiento y cumplimiento de los requisitos de la aplicación de teléfono móvil desarrollada. Los casos de prueba se diseñaron con el objetivo de evaluar tanto la funcionalidad como la eficacia del sistema en situaciones reales de uso. A continuación, se describen los casos de prueba utilizados y los resultados obtenidos:

5.1 Caso de prueba de medición de pulsaciones

El reloj tiene un sensor de fotopletismografía el cual mide los latidos por minuto. En el hipotético caso de que el reloj donde se cargue la aplicación no disponga de dicho sensor, aparecerá el icono de un corazón roto para indicar que no se ha podido acceder a dicho sensor o no lo tiene. Gracias a la siguiente función, la aplicación es capaz de acceder a los datos de salud, específicamente para la medición de frecuencia cardiaca:

```
private val measureClient = healthServicesClient.measureClient

suspend fun hasHeartRateCapability(): Boolean {
    val capabilities = measureClient.getCapabilitiesAsync().await()
    return (DataType.HEART_RATE_BPM in
capabilities.supportedDataTypesMeasure)
}
```

Código 4. Función de comprobación de disponibilidad del sensor de latidos

La función *suspend fun hasHeartRateCapability(): Boolean* es una función suspendida que verifica si el dispositivo tiene la capacidad de medir la frecuencia cardíaca a través de la API de Google Fit.

val capabilities = measureClient.getCapabilitiesAsync().await() llama al método *getCapabilitiesAsync()* del cliente de medición para obtener las capacidades de medición del dispositivo de manera asíncrona, y luego espera (*await()*) a que se complete la operación.

No se obtienen cada cierto tiempo, sino siempre que estén disponibles. Por lo tanto, siempre que esté libre el sensor para medir las pulsaciones, se podrán obtener.

Cuando se coloca el reloj en la muñeca existe la posibilidad que por motivos de calibración en los datos al inicio de unas pulsaciones dé 0, así que simplemente se ha puesto un condicional descartando los valores más pequeños que 35 ppm.

5.2 Caso de prueba de detección de somnolencia al volante

Para realizar un caso de prueba de detección de somnolencia al volante en una aplicación de smartwatch se deben considerar los siguientes aspectos:

1. Definir los criterios para detectar la somnolencia, la frecuencia cardíaca en este caso.

2. Verificar que la aplicación es capaz de recopilar los datos necesarios para detectar la somnolencia de forma precisa y en tiempo real. (realizado en los casos de prueba de medición de pulsaciones y conectividad)
3. Probar la aplicación en distintas situaciones de conducción, como carreteras rectas o sinuosas, en diferentes horarios del día y en distintas condiciones climáticas.
4. Comprobar que la aplicación es capaz de detectar la somnolencia, desde una somnolencia leve hasta una somnolencia profunda. La clave está en detectar esa somnolencia leve, ya que una fuerte somnolencia se detecta claramente en esta aplicación debido a la bajada significativa de los latidos.
5. Verificar que la aplicación es capaz de notificar, mediante una alerta sonora, al usuario en caso de detectar somnolencia.
6. Comprobar que la aplicación no genera falsos positivos, es decir, que no detecta somnolencia cuando el usuario está completamente despierto y alerta. Este punto es importante porque sabiendo que es complicado mantener un ratio de acierto elevado utilizando solamente sensores para medir latidos, se ha tenido que establecer unas condiciones más sensibles y pueden dar lugar a estos falsos positivos. Por este motivo, se considera mejor avisar con más frecuencia debido a estos errores que no avisar cuando se debería.
7. Verificar que la aplicación es capaz de funcionar de manera continua durante un período prolongado sin interrupciones. (realizado en los casos de prueba de rendimiento)

5.3 Caso de prueba de conectividad entre aplicación y smartwatch

El primer mensaje siempre lo envía el teléfono preguntando si está la aplicación abierta. La función *getNodes(context: Context): BooleanArray* se utiliza para obtener la lista de nodos conectados a la aplicación móvil, enviar un mensaje a cada nodo y esperar una respuesta de confirmación del reloj inteligente.

Para esperar la respuesta de confirmación del reloj inteligente, la función utiliza tres bucles condicionales. Cada bucle tiene una espera de tiempo (100ms, 250ms y 350ms respectivamente) y comprueba si se ha recibido la confirmación de respuesta esperada del reloj inteligente.

Una vez recibe el mensaje el reloj, si la ruta del mensaje es la misma que la ruta de un mensaje de "aplicación abierta" definido en la constante “APP OPEN WEARABLE PAYLOAD PATH”, se envía un mensaje de confirmación al dispositivo que envió el mensaje original para indicar que la aplicación está abierta.

Por ese motivo para realizar un caso de prueba de conectividad entre la aplicación y el smartwatch se deben considerar los siguientes aspectos:

1. Comprobar que el teléfono y el smartwatch están correctamente vinculados mediante Bluetooth.
2. Verificar que la aplicación es capaz de detectar el smartwatch correctamente.
3. Probar que la aplicación es capaz de enviar y recibir datos con el smartwatch.

4. Realizar pruebas en distintas situaciones de conexión, como estar cerca del teléfono o alejado, para comprobar que la conectividad no se ve afectada por la distancia, aunque se supone que en situaciones reales el conductor esté al lado de su teléfono y con el reloj bien atado en su muñeca.
5. Comprobar que la aplicación sigue funcionando correctamente después de haber estado conectada durante un tiempo prolongado.
6. Verificar que los datos enviados desde el smartwatch son procesados correctamente por la aplicación y que no hay pérdidas de información.

5.4 Caso de prueba de rendimiento

El caso de prueba de rendimiento se centrará en evaluar el impacto de la ejecución continua de la aplicación en el consumo de batería del dispositivo móvil y del smartwatch. Para ello, se llevarán a cabo las siguientes acciones:

1. Se realizará una prueba con la aplicación en ejecución, miden los datos continuamente durante un tiempo determinado de 2 horas, ya que es el tiempo en el cual recomiendan parar a descansar en un trayecto en coche.
2. Se registrarán la capacidad de batería teórica y el nivel de batería tanto del dispositivo móvil como del smartwatch antes de la prueba.
3. Durante la prueba, se mantendrá la pantalla encendida con un uso continuo de la aplicación.
4. Al finalizar la prueba, se medirá el nivel de batería restante en ambos dispositivos.
5. Se comparará el nivel de batería antes y después de la prueba para determinar si la ejecución continua de la aplicación y el envío de mensajes han afectado significativamente el consumo de batería.

Se considerará que la prueba ha sido satisfactoria si se obtiene una diferencia de consumo de batería mínima entre ambos dispositivos antes y después de la prueba. En caso contrario, se deberán revisar los procesos de la aplicación y optimizar su rendimiento para minimizar su impacto en el consumo de batería del usuario.

Dispositivo	Samsung Galaxy Z Flip 4	Samsung Galaxy Watch 5
Capacidad	3700 mAh	410 mAh
Batería Inicial	100 %	100 %
Batería Final	85 %	87 %
Porcentaje de bajada	15 %	13%

Tabla 2. Especificaciones Batería entre móvil y smartwatch

Después de hacer la prueba, se ha considerado satisfactoria ya que no se ha superado un consumo de batería de más de un 15 % en los dos casos.

Se puede observar que, aunque el reloj tenga menos capacidad, ha bajado menos porcentaje por su SO, que está optimizado para enviar datos BLE y medir las pulsaciones.

Siendo la aplicación de móvil la que se encarga de la carga de trabajo de almacenar los datos en la BD y mostrarlos gráficamente, es lógico pensar que consume más la app del móvil por su tamaño, brillo y calidad de la pantalla.

5.5 Ideas de mejora

Aunque los resultados no fuesen del todo precisos al utilizar solamente los sensores de fotopletismografía, se ha podido observar que con algo más de tiempo e ingenio se podría perfeccionar el algoritmo e incluso implementar ciertas funcionalidades para hacer más efectiva la aplicación:

- Una posible idea es almacenar información del usuario como edad, peso, sexo, etc. Para dar una solución más personalizada.
- Poder refinar el algoritmo de detección agregándole lecturas de los sensores del giroscopio del reloj para hacer un estudio de movimiento de la muñeca al conducir para predecir movimientos extraños, el GPS para ver si da algún acelerón en algún momento de somnolencia ocasional en el que no se da cuenta uno del peso del pie en el pedal, etc.
- Desde luego una posible implementación para detectar el sueño sería colocar el teléfono en algún soporte homologado para la conducción, que hoy en día muchas personas utilizan para seguir las rutas en apps de mapas, y así utilizar la cámara frontal del dispositivo para hacer un seguimiento del pestañeo del usuario mediante visión por computador y avisar al usuario cuando detecte que cierra los ojos más de lo debido o pestañea mucho, lo cual sería una señal de somnolencia y además de avisarle debería recomendarle parar a descansar. Esta idea ya se está pensando y poniendo a prueba donde explican en un informe de la DGT que entorno a un 30% de los accidentes de tráfico se relacionan de un modo u otro con la fatiga e intentan implementar en ADAS[20] [17].
- El hecho de implementar el GPS podría recomendarle parar cada cierto tiempo de trayecto y kilómetros recorridos en algún área de servicio cercana utilizando APIs de Google Maps, sin embargo, son de pago y no se ha podido llevar a cabo.
- También estaría bien tener una pestaña con una serie de estadísticas de los trayectos que ha ido haciendo el usuario como cuántos trayectos ha realizado, tiempo transcurrido, veces que ha tenido que alertarle la aplicación, un gráfico de la media de latidos por minuto en cada trayecto según la duración de este o la franja horaria para saber en qué momentos del día conduciendo tiene más probabilidades de dormirse.

[20] Advanced Driver Assistance System

6 Conclusiones

En conclusión, el desarrollo de una aplicación en Android para la detección de somnolencia al volante mediante el uso de un smartwatch representa un paso importante en la mejora de la seguridad vial. A través de la combinación de tecnología de vanguardia, como la detección de latidos gracias a los sensores del smartwatch y el procesamiento de datos en tiempo real, se ha logrado crear una solución innovadora que tiene el potencial de prevenir accidentes de tráfico causados por la somnolencia del conductor. El enlace al repositorio de este TFG, tanto la aplicación móvil como la aplicación para smartwatch se encuentra en GitHub. [18]

Durante el desarrollo de este trabajo, se han enfrentado diversos desafíos, como el acceso a los sensores del dispositivo, la comunicación entre móvil y reloj, el análisis y procesamiento de los datos, y la implementación del algoritmo de detección de somnolencia. Sin embargo, los resultados obtenidos demuestran el éxito en la creación de una aplicación funcional que puede alertar a los conductores sobre su nivel de somnolencia en tiempo real, lo que les permite tomar medidas preventivas y evitar situaciones de riesgo en la carretera.

El desarrollo de este trabajo fin de grado ha supuesto un reto personal y profesional para mí, ya que me ha permitido aplicar los conocimientos adquiridos a lo largo de la titulación de ingeniería informática, así como aprender nuevas habilidades y competencias relacionadas con el diseño y desarrollo de aplicaciones móviles, el uso de dispositivos wearables, el análisis de datos y cómo tratarlos, etc. Asimismo, he podido profundizar en un tema de gran relevancia social y seguridad vial, como es la prevención de accidentes por somnolencia al volante, y contribuir con una propuesta que podría mejorar la seguridad vial y el bienestar de los conductores.

Referencias

[1] DGT - Conducir con sueño o cansancio. [Accedido: 27 de febrero de 2023]

[2] Fatiga al volante, peligro constante (dgt.es). [Accedido: 03 de marzo de 2023]

[3] DGT - Conducir con sueño o cansancio. [Accedido: 03 de marzo de 2023]

[4] Copiloto | Samsung España. [Accedido: 12 de marzo de 2023]

[5] Drowsy Driving Alert - Aplicaciones en Google Play. [Accedido: 12 de marzo de 2023]

[6] Nap-Zapper | Mousetraps, insect traps, mosquito lights | Garden Tools | Non-powered Hand Tools | Hardware & Tools | CENS.com. [Accedido: 12 de marzo de 2023]

[7] Kotlin for Android | Kotlin Documentation (kotlinlang.org) . [Accedido: 20 de marzo de 2023]

[8] Google (2013) Android Studio (Versión: Electric Eel | 2022.1.1) (Programa de ordenador). Disponible en: Download Android Studio & App Tools - Android Developers. [Accedido: 20 de marzo de 2023]

[9] Room | Jetpack | Android Developers. [Accedido: 25 de marzo de 2023]

[10] Cómo enviar y sincronizar datos en Wear | Desarrolladores de Android | Android Developers. [Accedido: 15 abril de 2023]

[11] Por qué la luz azul no nos deja dormir (holadoctor.com). [Accedido: 29 de marzo de 2023]

[12] health-samples/health-services at main · android/health-samples · GitHub. [Accedido: 09 de abril de 2023]

[13] Servicios de salud en Wear OS | Desarrolladores de Android | Android Developers. [Accedido: 09 de abril de 2023]

[14] Monitorizar la frecuencia cardiaca con el Apple Watch - Soporte técnico de Apple (ES). [Accedido: 05 de abril de 2023]

[15] N.Zhang, M. Fard. M. U. H. Bhuiyan, D. Verhagen, M. F. Azari & S. R. Robinson. (Publicado: 06 de junio de 2018). “The effects of physical vibration on heart rate variability as a measure of drowsiness”, *Ergonomics*. Volumen: 61, Número: 9, Páginas: 1259-1272. DOI: https://doi.org/10.1080/00140139.2018.1482373

[16] Sang-Ho Jo, Jin-Myung kim & Dong Kyoo Kim. (Publicado: 04 de marzo de 2019). “Heart Rate Change While Drowsy Driving”, *Journal of Korean Medical Science*. Volumen: 34, Número: 8, Páginas: 5. DOI: https://doi.org/10.3346/jkms.2019.34.e56

[17] Detector de fatiga y sueño (dgt.es). [Accedido: 15 de abril de 2023]

[18] https://github.com/Bytheface1/DrivingAngel. [Accedido: 07 de junio de 2023]