Escuela Politécnica Superior

EarDrive

Desarrollo de un sistema de discriminación de alertas sonoras en la conducción para personas con discapacidad auditiva

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Francisco Laport López
Tutor/es:
Antonio Manuel Jimeno Morenilla

Universitat d'Alacant
Universidad de Alicante

Julio 2016

Resumen

Actualmente vivimos en un mundo donde los dispositivos inteligentes y las conexiones a internet están ocupando y accediendo a todos los aspectos de nuestras vidas, y donde ya no resulta extraño tener asistentes personales o comunicaciones instantáneas en nuestros teléfonos móviles.

Todos estos avances están orientados a facilitar y hacer más simple la vida de sus usuarios, pero este auge de desarrollo parece ir más retrasado cuando se trata de aplicaciones para personas con diversidad funcional. Pocas son las empresas que se aventuran a investigar e invertir en estos campos con un público objetivo tan reducido.

En este proyecto se ha llevado a cabo una aplicación móvil que ayuda en la conducción a personas con dificultades auditivas. La aplicación actúa como un oído artificial para el conductor, siendo capaz de capturar todos los sonidos que rodean al vehículo y de detectar aquellos que puedan representar una alerta auditiva. El sistema se comunica con el usuario a través de señales visuales y vibratorias. Se ha incluido un reloj inteligente dentro del sistema para que los avisos de la aplicación se reproduzcan también en él.

Justificación y objetivos

El trabajo de fin de grado de una ingeniería debe ser el reflejo de todos los conocimientos adquiridos por el estudiante durante su formación en la carrera. De esta forma, un ingeniero multimedia debería ser capaz de plasmar sus conocimientos en áreas tan complejas como el tratamiento de sonido, tratamiento de imagen, diseño y programación de aplicaciones tanto web como móviles o incluso en el campo del entretenimiento y ocio digital enfocado en los videojuegos. Además, un ingeniero multimedia ha de saber dirigir y organizar un proyecto y a sus respectivos grupos de trabajo, coordinando sus actividades y midiendo sus tiempos.

Debido a todo esto, he elegido este proyecto como trabajo final de grado no solo por el interés que me despierta el desarrollo de aplicaciones móviles, sino porque creo que es una buena oportunidad para aprovechar todos los conocimientos que he adquirido durante la universidad y crear una herramienta que facilite la vida y las tareas cotidianas a personas que presentan discapacidades auditivas, y aportar así mi granito de arena a poder hacer más accesible el mundo de las tecnologías y por lo tanto más justo y generalizado.

La mayoría de lo que se desarrolla hoy en día para plataformas móviles o webs está orientado a un perfil de usuario que no presenta ningún tipo de discapacidad (auditiva, visual, motora, etc.), por lo que he pensado que el desarrollo y la investigación de una aplicación enfocada a un perfil de usuario que sí presente discapacidades sería una buena forma de ampliar mis conocimientos de accesibilidad y poder explorar una parte de las tecnologías que, a mi parecer, se encuentran un poco apartadas de las líneas frecuentes de investigación.

Concretamente, el **objetivo principal** de este proyecto es el desarrollo de un sistema que ayude al usuario a detectar situaciones de peligro a través de alertas sonoras en la conducción. Como he explicado anteriormente, el público objetivo serán usuarios que tengan dificultades auditivas y que por lo tanto, el sistema actúe como un oído artificial que pueda escuchar y localizar todos aquellos ruidos que puedan representar un peligro para el conductor.

Este sistema se ha implementado como una aplicación móvil llamada ***EarDrive,*** con la cual el usuario podrá utilizar su teléfono móvil para que le avise de las alertas sonoras que se produzcan mientras este conduce. Además podremos conectar nuestra aplicación con un reloj inteligente y que estos avisos se reproduzcan en él también.

Agradecimientos

Quisiera agradecer en este documento a todas aquellas personas que han hecho posible el desarrollo de este Trabajo Final de Grado.

En primer lugar, gracias a todas y todos mis compañeras y compañeros de carrera que durante estos cuatro años han compartido la experiencia universitaria conmigo. Pero sobre todo, gracias a mis compañeros de grupo del ABP, por ayudarme y aguantarme en todo momento y hacer posible que viviese una experiencia inolvidable sin dejarme atrás.

Gracias también a mi tutor Antonio Manuel Jimeno Morenilla por su tiempo y por su dedicación, por sus aportes a este proyecto y por no poner nunca un impedimento, ayudándome siempre y apoyándome para avanzar. También agradecer a profesores como Diego Marcos, José Manuel Iñesta Quereda o Sergio Bleda, que, sin tener ninguna relación con este TFG, se prestaron muy amablemente a ayudarme e ilustrarme en aquellos campos que yo desconocía.

Por último, agradecer a mi familia y a mi pareja por su ayuda y sus ánimos, que, pese a la distancia, han sido de esencial importancia para el desarrollo de este proyecto.

Gracias a la vida.

Citas

"Todo es gozo cuando se pelea por la luz del mundo"

José Martí

Índice de contenidos

Índice de figuras

1 Introducción

Actualmente vivimos en un mundo donde los dispositivos inteligentes y las conexiones a internet están ocupando y accediendo a cada uno de los distintos aspectos de nuestras vidas, y donde ya no resulta extraño tener asistentes personales o comunicaciones instantáneas en nuestros teléfonos móviles.

Estos avances han permitido que los usuarios de estas tecnologías lleven una vida más fácil y simple, a través de la automatización de tareas, de la continua conexión a internet y su rápido acceso a la información o del mero hecho de conocer nuestra posición en el mapa. Pero todo este desarrollo parece no experimentar el mismo auge cuando se trata de aplicaciones móviles orientadas a usuarios con discapacidad auditiva.

Son pocas las empresas que se arriesgan a investigar y realizar aplicaciones tan específicas y con un público objetivo tan reducido. Cierto es, que en el mercado actual podemos encontrar tecnologías y distintos tipos de ayudas para personas con dichos problemas, pero muchas veces resultan de difícil acceso y excesivamente caros. Un claro ejemplo de esto es el 'Telecommunication Device for the Deaf' , dispositivo que permite la comunicación por teléfono a personas con dificultades auditivas, y cuyo precio en su gama más baja supera los 200 dólares.

Según el informe "Sociedad en red" publicado por el Ministerio de Industria, Energía y Turismo de España, la penetración de los *smartphones* en la sociedad española supera el 53% en usuarios mayores de 15 años **[1].** Es decir, más de la mitad de la población mayor a 15 años de edad posee un Smartphone, por lo que no cabe duda de que nuestro sistema ha de ser desarrollado para este tipo de dispositivos, ya que su uso es generalizado y crece de manera exponencial.

De esta forma, un sistema que funcione sobre un dispositivo móvil inteligente aumentará su accesibilidad y eliminará casi al mínimo las barreras económicas, ya que como acabamos de analizar, un gran porcentaje de la población posee un Smartphone y lo utiliza en su día a día para sus tareas más frecuentes.

Además de a nuestros teléfonos móviles, la tecnología y sus avances también han llegado a dispositivos tan cotidianos como los relojes, dando lugar a los Smartwatch, los cuales permiten al usuario controlar diferentes funcionalidades de su Smartphone y recibir ciertas notificaciones. Actualmente su uso no está muy extendido, pero se espera que en unos años éste sea un complemento indispensable para nuestro teléfono inteligente. Es por esto por lo que las grandes compañías y las aplicaciones exitosas han comenzado a dar soporte a este tipo de tecnologías.

Hoy en día, la tecnología está suficientemente desarrollada como para dar soluciones a personas con discapacidad auditiva a través de apps móviles. El concepto Realidad Aumentada a menudo relacionado con la percepción visual, puede extenderse al mundo auditivo, convirtiendo señales sonoras no audibles por algunas personas en información reconocible con el fin de aumentar su sensación de inmersión en la realidad.

2 Marco Teórico

El proyecto desarrollado se ha construido en base a una cantidad amplia de conceptos, proyectos relacionados con el sector y diferentes librerías y funcionalidades de la API de Google. En este apartado pasaremos a analizar cada una de ellas y a explicar cuál es el papel y su importancia dentro de la aplicación que se ha implementado.

2.1 La Discapacidad Auditiva y la conducción

La Organización Mundial de la Salud (OMS) define como discapacidad auditiva (DA) a la pérdida auditiva superior a 25dB, dentro de este concepto también se incluyen la hipoacusia[1], la sordera y la sordera profesional.

Una pérdida significativa de la audición puede afectar al rendimiento y la seguridad en la conducción. Suele acompañarse de restricciones en la actividad cotidiana, afectando las habilidades de comunicación y de interacción con el medio, teniendo importantes repercusiones físicas, psicológicas y económicas.

En 2009, la OMS estimaba que a nivel mundial el número de personas con DA era de 278 millones. En Europa, cerca de 71 millones de adultos entre los 18 y 80 años tienen pérdida auditiva, y en España se estima que cerca del 10% de la población entre 6 y 65 años tiene algún tipo de DA.

El grupo con mayor probabilidad de tener una discapacidad auditiva es aquel que se encuentra en una edad superior a los 65 años, siendo esta la enfermedad crónica más frecuente. Por otra parte, los jóvenes también pueden presentar casos de DA, ya que la audición se comienza a perder a partir de los 20 años.

La discapacidad auditiva la podemos clasificar según el grado de pérdida de audición:

- **Leves**: El umbral auditivo se encuentra entre 41-71dB.
- **Medias**: La pérdida auditiva se encuentra entre 41-70 dB.

[1] **Hipoacusia:** Es la incapacidad total o parcial para escuchar sonidos en uno o ambos oídos.

- **Graves**: La pérdida auditiva se encuentra entre 71-90 dB.
- **Profundas**: La pérdida auditiva supera los 90dB y se sitúa entre 91-100 dB.

1.1.1 Normativas de la DGT en relación de la Discapacidad Auditiva

La normativa exige que para conducir no debe existir una pérdida de audición combinada entre los dos oídos, con o sin audífono, de más del 45% para el grupo 1 (AM, A1, A, B, B+E y LCC) y de más del 35% para el grupo 2 (BTP, C1, C1+E, C+E, D1, D1+E, D, D+E) obteniendo el índice de esta pérdida mediante la audiometría.

Además, cuando se presenta un déficit sensorial, la DGT obliga a realizar un mecanismo compensatorio basado en potenciar la capacidad de otras aéreas sensoriales, tales como la visión. En el caso de la hipoacusia es obligatoria la utilización de espejos retrovisores exteriores y uno panorámico en el interior del coche. De esta manera se aumenta el campo visual del conductor.

También establece que no deben existir alteraciones del equilibrio (vértigo, inestabilidad, mareo, vahído) permanentes, evolutivas o intensas. **[2]**

2.2 Aplicaciones relacionadas con la audición

En este apartado se procederá a explicar las aplicaciones que se han encontrado en el marcado que presentan relación con el proyecto a realizar.

2.2.1 Audio Aware

El funcionamiento de la aplicación, que se ejecuta en segundo plano en un móvil Android (inicialmente) es muy sencillo: la app tiene registrado una serie de sonidos peligrosos (por ejemplo chillidos o sirenas), es capaz de asociarlos con los que ocurren en la realidad, y en ese alertar a los usuarios, interrumpiendo la música del teléfono o amplificando la señal de alerta.

Este sistema puede ser de gran utilidad para personas que tienen dificultades de audición (se puede amplificar el sonido de peligro detectado), pero también para aquellos usuarios que van por la calle despistados, y pueden estar en peligro. **[3]**

2.2.2 My Ear Droid

Aplicación para la detección e identificación de los sonidos habituales que se generan en el hogar. En su versión para móvil, es una aplicación Android que avisa o alerta al usuario de sonidos que se producen en su entorno. De esta manera, utilizando su propio Smartphone, una persona con discapacidad auditiva podrá estar informada de eventos sonoros como timbre, teléfono, alarma de incendios, etc. **[4]**

Únicamente reconoce sonidos característicos del hogar y que el usuario haya grabado previamente. La aplicación está disponible en la Play Store gratuitamente para cualquier usuario.

2.2.3 BioAid

Aplicación móvil que amplifica por los auriculares todos aquellos sonidos que escucha. Es una aplicación muy útil para aquellas personas que tienen alguna dificultad auditiva y todavía no llevan un audífono. **[5]**

Tras analizar las aplicaciones ofertadas en el mercado para personas con discapacidad auditiva, se puede concluir que, aunque se está comenzando a desarrollar para este público objetivo, es todavía muy reducido el número de apps que se pueden encontrar. Además, dentro de este pequeño mercado, no existen aplicaciones que tengan como objetivo servir de ayuda para la conducción a personas con un problema auditivo, por lo que nuestra aplicación podría tener una gran acogida por los usuarios.

2.3 Desarrollo de apps sobre tecnologías móviles

Para la implementación de este proyecto se ha realizado un estudio de las distintas plataformas que nos permiten desarrollar hoy en día aplicaciones para móviles, teniendo en cuenta que la app que se desea obtener debe cumplir los principios de universalidad, eficiencia y bajo coste. Es decir, se pretende que el sistema llegue al mayor número de

personas, que permita una ejecución fluida en la mayoría de dispositivos y que su coste no sea excesivamente caro.

2.3.1 iOS

iOS es un sistema operativo móvil de la multinacional Apple Inc. Originalmente desarrollado para el iPhone (iPhone OS), después se ha usado en dispositivos como el iPod touch, el iPad y el AppleTV. Es el segundo sistema operativo móvil más utilizado en el mundo, llegando a liderar en mercados tan importantes como el chino o el japonés. En España su cuota de mercado crece año tras año, como podemos observar en la figura 1, pero la diferencia que presenta con Android es todavía abismal.

iOS posee una interfaz fluida, sencilla y elegante, sin mucha posibilidad de personalizar pero que ofrece al usuario una de las experiencias más cómodas del mercado. Esto se debe a que iOS está diseñado para sacar el máximo provecho al Hardware que coloca en sus dispositivos el cual siempre se ha diferenciado considerablemente de los demás fabricantes. **[6]**

Su simplicidad y optimización son sus pilares básicos para que millones de usuarios se decanten por iOS en lugar de escoger otras plataformas que necesitan más hardware para mover con fluidez el sistema. **[7]**

Estas características encajan a la perfección con los objetivos del proyecto, pero analizando los precios de los dispositivos que funcionan con este sistema operativo se deduce que son más elevados que los de sus competidores. Si a esto se suma que no es el SO móvil más utilizado, se puede concluir que esta elección se contrapondría al principio de universalidad y bajo coste, ya que estos teléfonos móviles no están al alcance de la mayoría ni son los más utilizados. Esta es la principal razón por la que se ha descartado este sistema operativo en este proyecto.

2.3.2 Windows Phone

Windows Phone es un sistema operativo móvil desarrollado por la empresa Microsoft para teléfonos inteligentes y otros dispositivos móviles. Fue lanzado al

mercado el 21 de octubre de 2010 en Europa y el 8 de Noviembre en Estados Unidos, con la finalidad de suplantar el conocido Windows Mobile.

Microsoft decidió realizar un cambio completo en este nuevo sistema operativo con respecto al otro, no solo se cambió el nombre, sino que se desarrolló desde cero, presentando una interfaz completamente nueva, mejor comportamiento y un mayor control sobre las plataformas de hardware que lo ejecutan, todo con el propósito de volver a ser competitivo en el mundo de los móviles. **[8]**

Este sistema operativo destaca por su fluidez y su rendimiento, siendo en muchos casos superior a Android. Mientras el SO líder necesita un hardware de gama media-alta para ofrecer un rendimiento bueno, con Windows Phone obtendremos los mismos resultados con un hardware de inferiores prestaciones. Además, cabe destacar que este rendimiento combinado con su simplicidad en el diseño de las interfaces proporciona una gran experiencia de usuario.

Pese a estas ventajas, nos encontramos con un problema similar a iOS respecto al principio de universalidad. Si se observa la figura 1 se deduce que la cuota de mercado de Wndows Phone en España es muy reducida, y que cada año va perdiendo usuarios. Por lo que optando por este sistema estaríamos reduciendo mucho el público objetivo y por lo tanto perdiendo posibles beneficiarios de la aplicación.

2.3.3 Android

Android es actualmente el sistema operativo con mayor número de usuarios en el mundo, ya sea para dispositivos móviles, tablets, o phablets. Es un sistema basado en Linux, un núcleo de sistema operativo libre, gratuito y multiplataforma.

De esta forma, desarrollando nuestro sistema para dispositivos móviles eliminaríamos una de las barreras principales de las tecnologías actuales orientadas a personas con diversidad funcional, la barrera económica, ya que como hemos analizado antes, un gran porcentaje de la población ya utiliza un Smartphone en su día a día. Con esto se conseguirá una mayor accesibilidad y por lo tanto que un número más elevado de usuarios puedan sacar partido a la aplicación.

Como podemos observar en la siguiente figura, en España el sistema operativo para móviles predominante es Android, por lo que la aplicación deberá ser totalmente compatible con dicho SO. Además este sistema operativo cuenta con una comunidad de desarrolladores muy amplia y muy activa, por lo que existe una extensa documentación en la que basar el desarrollo. Este hecho, garantiza el principio de universalidad que se ha expresado anteriormente, ya que el público potencial es el mayor con diferencia en comparación con otros sistemas.

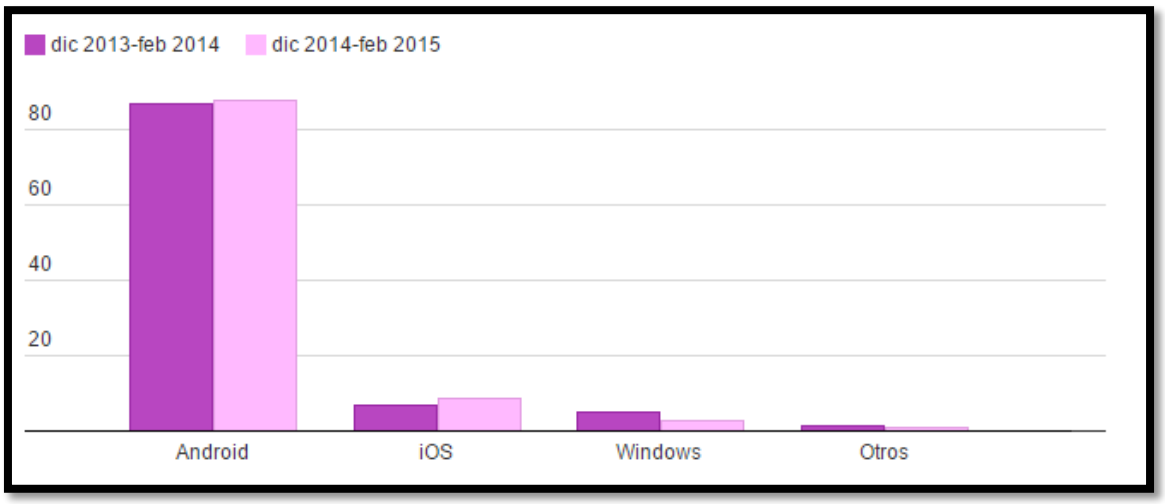


Figura 1 - Ventas de SmartPhones según su SO en España

El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla en un lenguaje de programación muy conocido como es Java.

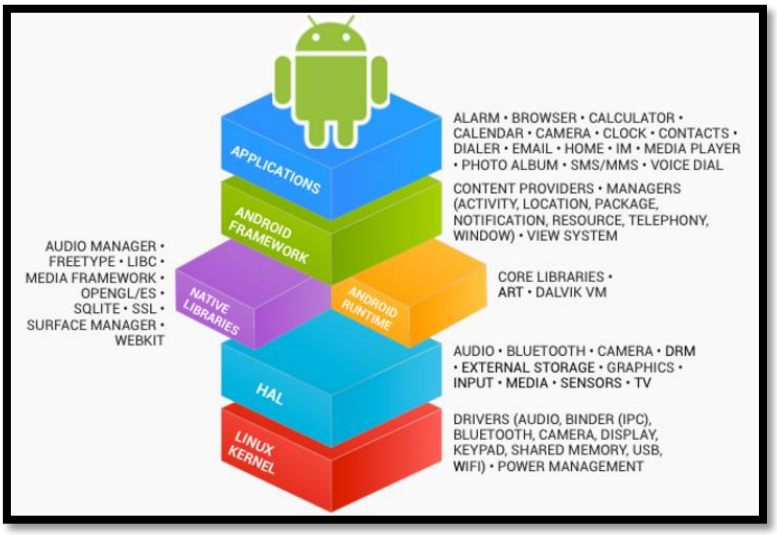


Figura 2 – Android Stack

Fuente: https://source.android.com/source/index.html

Una de las características más importantes de Android es que es un sistema operativo libre, no debemos pagar nada ni para desarrollar para él ni para instalarlo en los dispositivos, lo que concuerda perfectamente con el principio de bajo coste del proyecto. Además, su código fuente está abierto a toda la comunidad de desarrolladores, por lo que los fabricantes de Smartphones o desarrolladores pueden adaptar dicho sistema a su gusto y funcionamiento. **[9]**

Para facilitar la expansión y generalizar el uso de este sistema operativo, Android mantiene una página web orientada a sus desarrolladores, donde podemos encontrar una amplia y profunda documentación de todas las funcionalidades del sistema así como de la API. Aquí también encontraremos una guía con todos los pasos que debemos seguir para publicar nuestra aplicación en la tienda de Google Play, empezando por el diseño, siguiendo por el desarrollo y terminando por la distribución y monetización de la app.**[10]**

Actualmente la última versión de Android es Marshmallow (Android 6.0) y su versión 23 de la API. En esta última actualización del sistema operativo se han incluido determinadas funciones que debemos tener en cuenta a la hora de desarrollar la aplicación:

- **Permisos en tiempo de ejecución:** En las versiones anteriores de Android dábamos permisos a las aplicaciones a la hora de instalarlas. Con la llegada de Android Marshmallow se ha introducido un nuevo modelo de permisos. Estos serán requeridos al usuario cuando la aplicación sea ejecutada, y éste tendrá en todo momento acceso y capacidad para activarlos o desactivarlos.
- **Doze y ahorro de batería:** Se trata de una evolución del modo ahorro de batería en el que podemos elegir una a una qué aplicaciones funcionarán a pleno rendimiento de manera optimizada y cuáles inactivas ahorrando batería. **[11]**. Además, se ha añadido **App StandBy** el cual determina cuando una aplicación lleva mucho tiempo sin usarse y que el usuario no está siendo activo en ella y desconecta automáticamente todas las conexiones a la red y suspende las tareas de dicha aplicación.
- **Eliminación de Apache Client HTTP:** En la versión 6.0 de Android se ha eliminado el soporte para Apache HTTP client, por lo que se debe hacer uso de las clase HttpURLConnection.
- **Cambios en el AudioManager:** Con esta nueva versión de Android ya no será posible ajustar el volumen o silenciar un audio a través del AudioManager. Estas opciones han sido eliminadas y sustituidas por otras como por ejemplo adjustStreamVolume().

Como podemos observar en la siguiente figura, a la última versión de Android todavía le queda mucho para ser la más utilizada, este puesto lo tiene Android Lolipop, la versión anterior a Marshmallow. El segundo puesto lo lleva Android Kit-Kat seguido de Android Jelly Bean.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.3%
4.1.x	Jelly Bean	16	8.1%
4.2.x		17	11.0%
4.3		18	3.2%
4.4	KitKat	19	34.3%
5.0	Lollipop	21	16.9%
5.1		22	19.2%
6.0	Marshmallow	23	2.3%

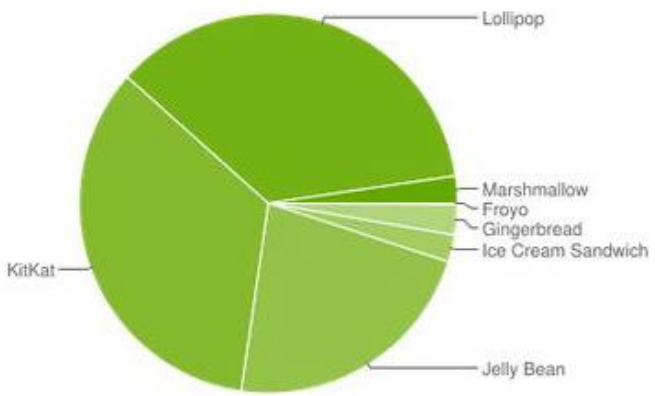


Figura 3 - Uso de las distribuciones de Android

Fuente: http://andro4all.com/2016/03/lollipop-android-mas-utilizado-marshmallow-menos

A la hora de realizar la aplicación debemos tener muy en cuenta todos estos datos, ya que si desarrollamos para una versión de la API muy nueva dejaremos a un gran número de posibles usuarios fuera de nuestro alcance. Lo apropiado sería que la aplicación fuese compatible con todas las distribuciones del sistema, pero si esto no resulta posible, debemos intentar que por lo menos lo sea con aquellas que tienen un número muy elevado de usuarios, como por ejemplo Jelly Bean.

2.3.4 Android Wear

Android Wear es el sistema operativo para dispositivos corporales (wearables) basado en Android que Google presentó a la sociedad el 18 de marzo de 2014. El sistema en sí está pensado para ser utilizado en relojes inteligentes (smartwatches), pulseras inteligentes (smartbands), y cualquier otro dispositivo wearable que pueda surgir en el futuro. **[12]**

La aplicación para SmartWatches se desarrollará para este SO, ya que resulta totalmente compatible con Android y que su implementación es muy similar, por lo que el tiempo de aprendizaje será rápido y corto.

Además, Android Wear es el sistema operativo más usado en el mundo de los wearables, por lo que resulta la mejor elección si queremos que la aplicación tenga un número elevado de usuarios.

Actualmente existen varias tecnologías móviles con las que desarrollar aplicaciones e incluso se podrían utilizar tecnologías y lenguajes de programación web para su implementación, obteniendo así una aplicación híbrida. Pero debido a la complejidad de analizar señales sonoras en nuestro dispositivo he optado por desarrollar el proyecto en Android nativo, es decir, en Java.

2.4 Lenguajes de programación para apps móviles

Casi en cualquier ámbito de programación existe una amplia gama de lenguajes y entornos de desarrollo con los que trabajar, las aplicaciones móviles no constituyen una excepción. Se han analizado distintas opciones dentro de la plataforma tecnológica elegida (Android) y finalmente se ha optado por Java como se describe en el apartado siguiente.

2.4.1 Java

Java es un lenguaje de programación de propósito general, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados**. [13]**

Actualmente Java es el lenguaje más utilizado para la programación de las aplicaciones de Android, siendo este el lenguaje recomendado por sus desarrolladores. La propia IDE de Android (Android Studio), genera automáticamente los proyectos con dicho lenguaje, además, toda la documentación oficial viene explicada también en Java.

Se ha considerado que programar Android en su lenguaje 'nativo' y del que más documentación existe sería la opción más adecuada. La principal razón es que el sistema a desarrollar necesitará acceder a elementos del hardware del dispositivo y de esta manera resultará más fácil y sencillo. Por otra parte, Android nativo proporciona a los desarrolladores una serie de APIs y clases que facilitarán mucho la tarea del muestreado y filtrado de la señal sonora.

Finalmente decir que Java también tiene sus inconvenientes de los cuales quizás será la eficiencia uno de los más notables, sin embargo se ha considerado que los beneficios que aporta respecto a la facilidad de programación y su universalidad, tienen más peso en el contexto de este proyecto que sus debilidades .

2.4.2 XML

XML es un Lenguaje de Etiquetado Extensible muy simple, pero estricto que juega un papel fundamental en el intercambio de una gran variedad de datos. Es un lenguaje muy similar a HTML pero su función principal es describir datos y no mostrarlos como es el caso de HTML. XML es un formato que permite la lectura de datos a través de diferentes aplicaciones. XML sirve para estructurar, almacenar e intercambiar información. **[14]**

En las aplicaciones Android, XML juega un papel muy importante, con este lenguaje describiremos como serán las vistas de las distintas actividades de nuestra aplicación. Además, es utilizado para almacenar todo tipo de valores como por ejemplo colores, cadenas de texto, estilos o animaciones.

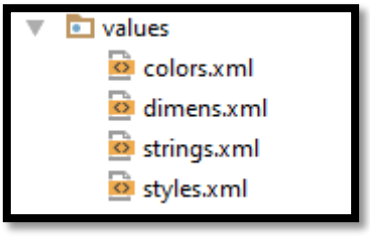


Figura 4 - Archivos XML en un proyecto Android

Fuente: Creación propia

3 Objetivos

El **objetivo principal** de este proyecto es la realización de una aplicación móvil que **ayude a las personas con discapacidad auditiva en la conducción**. El teléfono móvil ha de escuchar el sonido ambiente del vehículo y detectar aquellos sonidos que puedan resultar un peligro para el conductor, avisándole a través de una alerta visual y vibratoria.

Además, dentro del proyecto podemos encontrar diferentes objetivos específicos, los cuales están enfocados a cumplir con los principios de diseño que se han determinado para esta aplicación: universalidad, eficiencia y bajo coste.

3.1 Objetivos computacionales

- **Tiempo de respuesta de la aplicación**: Garantizar una respuesta rápida y efectiva. Es una característica muy importante del sistema. El tiempo que la aplicación tarde en responder a las interacciones del usuario o del medio debe ser muy corto, ya que el tiempo de reacción medio de las personas en la conducción es de 0,75 segundos por lo que si la aplicación tarda en detectar los peligros podría ser tarde para evitarlos.
- **Tiempo de comunicación con el SmartWatch**: Garantizar una comunicación inmediata o casi inmediata con el dispositivo wearable. Al igual que el tiempo de respuesta, el tiempo que tarde una alerta en llegar al reloj es de primordial importancia en el sistema. La comunicación entre ambos dispositivos ha de ser lo más rápida posible, intentando que llegue a ser instantánea para poder así evitar posibles peligros.
- **Tiempo de ejecución de algoritmos**: Garantizar un tiempo de ejecución rápido y óptimo. Algoritmos complejos como son el cálculo de la Transformada Rápida de Fourier (FFT) y el muestreo de la señal han de ser lo más óptimos posibles, ya que esto ayudará a mejorar el tiempo de respuesta de la aplicación.
- **Peso de la aplicación**: Minimizar el peso de la aplicación al máximo posible. Tanto la aplicación para Smartphone como la aplicación para el dispositivo

wearable han de pesar lo menos posible, evitando así problemas de descarga y memoria a los usuarios.

3.2 Objetivos Tecnológicos

- **Desarrollo aplicación para wearables:** Desarrollar una aplicación compatible con relojes inteligentes. Una parte importante de este proyecto es que la aplicación móvil sea capaz de comunicarse con un dispositivo wearable si el usuario lo desea. Para esto se ha de desarrollar otra aplicación para esta plataforma.
- **Compatibilidad con Android:** Garantizar la compatibilidad con el sistema operativo Adnroid. La aplicación ha de ser totalmente compatible y funcional con Android. Este es el SO más usado en los Smartphones, Tablets y Phablets, por lo que el número de posibles usuarios será mayor que con cualquier otro sistema operativo.
- **Compatibilidad con Android Wear:** Garantizar la compatibilidad con Android Wear. La aplicación ha de ser totalmente compatible con el sistema operativo para wearables de Android ya que es este el sistema operativo más usado en este tipo de dispositivos.

3.3 Objetivos del diseño de la aplicación

- **Usabilidad:** Obtener un diseño simple y fácil de reconocer y entender para el usuario. El diseño de la interfaz gráfica y su presentación al usuario final es una parte de mucha relevancia dentro del proyecto. La interfaz será la forma de comunicarse que tendrá la aplicación con un usuario que estará conduciendo un vehículo, por lo tanto esta interfaz ha de ser muy **intuitiva** y **sencilla**, que con un simple vistazo podamos reconocer lo que el sistema nos quiere decir.
- **Interferencia mínima en la conducción:** No desviar la atención a no ser que sea absolutamente necesario es imprescindible cuando se está conduciendo, por lo que se evitará en lo posible que la aplicación emita información cuando no sea necesario.

- **Adaptable a distintos dispositivos:** Conseguir que el diseño sea capaz de adaptarse a todos los dispositivos móviles y wearables que el usuario pueda utilizar, tanto atendiendo a su resolución como a su forma (redonda o cuadrada) y disposición (landscape o portrait).

3.4 Objetivos de distribución

Para dar a conocer la aplicación y conseguir el mayor número de usuarios y descargas se utilizarán las redes sociales.

- **Facebook:** Actualmente es la red social más utilizada en el mundo, por lo que resulta una herramienta muy útil para la difusión del proyecto. En ella se creará una página con el nombre del producto (***EarDrive)*** a la que se subirán las últimas noticias y avances del proyecto.
- **Twitter:** Es otra de las redes sociales más utilizadas hoy en día. Se creará una cuenta con el nombre de la aplicación y se irán tuiteando todos los avances y novedades del proyecto.

Además, para facilitar la distribución de la aplicación se creará una cuenta en la tienda oficial de Android, Play Store, y se subirá la aplicación. Allí los usuarios podrán encontrarla fácilmente y descargarla con toda confianza.

4 Metodología

En este apartado se explicará la forma de gestionar el proyecto, así como los repositorios y las copias de seguridad realizadas y también la metodología utilizada para la realización de la aplicación.

4.1 Metodología del desarrollo software

Para el desarrollo de este proyecto se seguirán unos tiempos y unas pautas propios de las metodologías ágiles para el desarrollo de software. La principal característica de este tipo de metodología es la entrega frecuente de software funcional, intentando realizar una entrega cada dos o tres semanas. Por otra parte, es una metodología que admite cambios en las funcionalidades y los requisitos del sistema, incluso en las etapas más tardías del desarrollo, por lo que se adapta a la perfección al proyecto implementado en este Trabajo Final de Grado.

Para poder conseguir estas entregas frecuentes debemos organizar el proyecto en diferentes iteraciones, y cada una de estas iteraciones estará dividida en diferentes etapas. Al final de cada iteración se debe entregar un software funcional que aporte valor al proyecto. Las iteraciones de dividirán de la siguiente forma:

Figura 5 - Etapas de una iteración

Fuente: Diapositivas de la asignatura Análisis y Especificación de Sistemas Multimedia

4.2 Control de versiones y repositorio

Para el control de las distintas versiones del proyecto así como para tener copias de seguridad por si en algún momento fallase la copia local se utilizará la herramienta Git. Además, utilizaremos GitHub como plataforma donde alojar este repositorio.

Por otra parte, se hará uso de la propia ayuda que facilita Android Studio para el control de versiones, permitiendo las operaciones pull, commit y push desde el propio entorno de desarrollo.

4.3 Gestión del proyecto

Para la planificación y organización del proyecto se ha utilizado una herramienta web llamada Trello. En ella se han organizado las distintas tareas que se deben implementar en tarjetas, las cuales a su vez se encontrarán dentro de listas que indicarán cual es la tarea general. Este proyecto tiene cuatro listas principales: **Diseño de la Interfaz Gráfica**, **Funcionalidades App, Análisis del sonido y las pruebas que se deben realizar**.

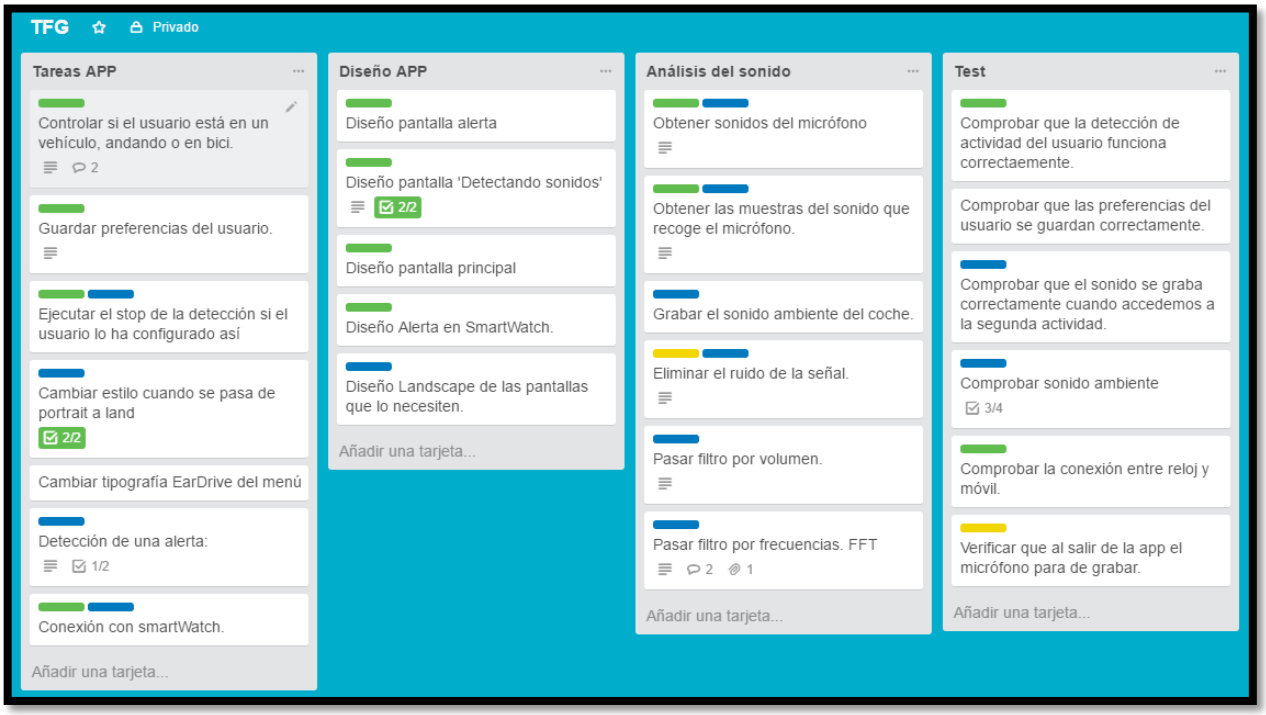


Figura 6 - Vista de la organización del proyecto en Trello

Fuente: Creación propia

Además, cada una de las tarjetas cuenta con una etiqueta la cual nos indicará cual es el estado de la tarea: Completada, En curso, Pausada o Descartada.

5 Especificación y análisis de requisitos

5.1 Requisitos funcionales

Mediante estos requisitos se describirán las funcionalidades que el sistema debe presentar. Se estructurarán siguiendo las directrices marcadas por el estándar IEEE 830,199.

Número de requisito	RF 01
Nombre de requisito	Capturar el sonido que llegue al micrófono
Tipo	Requisito
Prioridad del requisito	Alta / Esencial
Descripción	El sistema ha de ser capaz de capturar todos los sonidos que lleguen al micrófono del dispositivo.

Número de requisito	RF 02
Nombre de requisito	Habilitar la opción de parar de detectar peligros automáticamente.
Tipo	Requisito
Prioridad del requisito	Media
Descripción	El usuario podrá activar o desactivar la opción de que el sistema pare de detectar sonidos peligrosos cuando se detecte que dicho usuario está quieto o fuera de un vehículo.

Número de requisito	RF 03
Nombre de requisito	Habilitar la alerta vibratoria
Tipo	Requisito
Prioridad del requisito	Media
Descripción	El usuario podrá activar o desactivar la opción de que el sistema le avise cuando se produce una alerta a través de vibración.

Número de requisito	RF 04
Nombre de requisito	Habilitar la opción de enviar alertas al SmartWatch
Tipo	Requisito
Prioridad del requisito	Alta
Descripción	El usuario podrá activar o desactivar la opción de que el sistema envíe también las alertas a un dispositivo wearable.

Número de requisito	RF 05
Nombre de requisito	Conexión SmartWatch
Tipo	Requisito
Prioridad del requisito	Media

Descripción	Existirá un apartado en la aplicación donde el usuario podrá consultar si el móvil y la aplicación están conectados a un dispositivo wearable.

Número de requisito	RF 06
Nombre de requisito	Detectar sonidos peligrosos
Tipo	Requisito
Prioridad del requisito	Alta / Esencial
Descripción	El sistema ha de ser capaz de detectar aquellos sonidos que representen un peligro para el conductor.

Número de requisito	RF 07
Nombre de requisito	Detección del sonido ambiente
Tipo	Requisito
Prioridad del requisito	Alta
Descripción	El sistema ha de ser capaz de medir el sonido ambiente del vehículo y detectar si en él se está manteniendo una conversación o se está escuchando música, para así no generar falsas alertas.

Número de requisito	RF 08
Nombre de requisito	Parar de detectar sonidos

Tipo	Requisito
Prioridad del requisito	Alta / Esencial
Descripción	El sistema ha de ser capaz de parar la detección de sonidos cuando el usuario se lo indique.

Número de requisito	RF 8.1
Nombre de requisito	Para de detectar sonidos automáticamente
Tipo	Requisito
Prioridad del requisito	Alta / Esencial
Descripción	Si el usuario lo ha configurado, el sistema deberá detectar cual es la actividad del usuario (quieto, en coche, en bici,..) y parar la detección de sonidos cuando se encuentre quieto.

Número de requisito	RF 09
Nombre de requisito	Mostrar alerta
Tipo	Requisito
Prioridad del requisito	Alta / Esencial
Descripción	El sistema debe mostrar una alerta visual y vibratoria, si así lo ha configurado el usuario, cuando se detecte algún sonido peligroso. La alerta debe mostrarse por un tiempo (3 segundos) y luego desaparecer automáticamente.

Número de requisito	RF 9.1
Nombre de requisito	Mostrar alertas en SmartWatch
Tipo	Requisito
Prioridad del requisito	Alta
Descripción	Si el usuario lo ha configurado, el sistema deberá mostrar las alertas en el SmartWatch que se encuentre emparejado con el teléfono móvil. Estas alertas serán vibratorias y visuales, sin opción de configuración.

Número de requisito	RF 10
Nombre de requisito	Guardar las configuraciones del usuario
Tipo	Requisito
Prioridad del requisito	Media
Descripción	El sistema ha de ser capaz de almacenar las preferencias del usuario. Cuando el usuario salga y vuelva a entrar en la aplicación, estas preferencias han de estar como se configuraron por última vez.

5.2 Requisitos no funcionales

Con estos requisitos se describirán las propiedades que el sistema debe tener en su conjunto. Su estructura y presentación siguen las directrices marcadas por el estándar IEEE 830,199.

Rendimiento

El sistema ha de tener un alto rendimiento, transformándolo en una rápida respuesta al usuario en todas sus interacciones. Se deben optimizar todas aquellas tareas que puedan incrementar este tiempo de respuesta, como puede ser el análisis de las señales sonoras o la detección de las actividades que realiza el usuario. El objetivo del sistema ha de ser que las respuestas sean inmediatas.

Usabilidad

El sistema ha de ser usable y fácilmente comprensible para el usuario. La interfaz gráfica ha de ser intuitiva y simple, limitando así el tiempo de aprendizaje del uso de la aplicación.

La interacción usuario – aplicación será exclusivamente táctil, evitando utilizar métodos de comunicación por voz o sonoros.

Fiabilidad

El sistema ha de estar preparado para manejar y controlar todos los fallos y acciones que el usuario pueda cometer, avisándole de estos o simplemente controlándolos para que la aplicación no detenga su ejecución. Por ejemplo, si el usuario decide dejar en un segundo plano la aplicación, el sistema ha de ser capaz de pausar todos los procesos y liberar los recursos para garantizar el correcto funcionamiento del resto de aplicaciones.

Por otra parte, el sistema ha de ser fiable detectando alertas y peligros sonoros. Se debe reducir al máximo el número de falsas alertas, dotándose así de una mayor confianza hacia el usuario.

Mantenibilidad

Para el mantenimiento del sistema se deben tener unas prácticas de programación limpias y un patrón constante a seguir. Las líneas de código deben estar comentadas correctamente aclarando en todo momento cuál es su función. De esta

manera se reducirá de una forma considerable el tiempo invertido en el arreglo de fallos o bugs que se encuentren en el sistema.

Por otra parte, el sistema ha de estar modularizado, facilitando así que en versiones futuras se amplíen las funcionalidades.

Se debe además, revisar el sistema periódicamente y sobre todo el algoritmo de análisis de señales sonoras. Intentando encontrar en estas revisiones formas más optimas de ejecución así como incluyendo nuevos sonidos que puedan representar un peligro para los conductores.

Portabilidad

El sistema será compatible con versiones superiores o iguales a la 2.3 de Android (Gingerbread), abarcando así más del 98% de los teléfonos móviles Android. Si el sistema quiere ser usado con un dispositivo wearable, la versión de Android debe ser superior a la Kit-Kat.

La aplicación para SmartWatch será compatible con el sistema operativo Android Wear.

Por otra parte, la descarga del sistema será totalmente gratuita desde la tienda oficial de Android para aplicaciones.

5.3 Requisitos Hardware y Software

Para poder hacer uso de la aplicación *EarDrive* serán necesarios un software y un hardware con unas características determinadas:

5.3.1 Hardware

- **Teléfono móvil inteligente (Smartphone):** será necesario un Smartphone para poder ejecutar la aplicación. Para que el funcionamiento del sistema sea el adecuado se debe utilizar como mínimo un teléfono móvil de gama media.
- **Micrófono:** El teléfono móvil ha de llevar incorporado un micrófono que funcione correctamente para poder así escuchar los sonidos en la conducción.

- **Reloj inteligente:** no es esencial para el funcionamiento del sistema, pero es recomendable para mejorar la experiencia de usuario. El dispositivo debe ser capaz de soportar el sistema operativo Android Wear.
- **Conectividad Bluetooth:** Se necesitará una tarjeta Bluetooth para conectar el dispositivo móvil con el reloj inteligente.

5.3.2 Software

- **Sistema operativo Android:** el sistema operativo del teléfono móvil ha de ser Android con una versión igual o superior a la 2.3 (Gingerbread). Si se desea que el sistema funcione con un reloj inteligente entonces la versión de Andoird debe ser la 4.3 o superior.
- **Sistema operativo Android Wear:** si el sistema está funcionando con un reloj inteligente integrado, su sistema operativo ha de ser Android Wear para que funcione correctamente.
- **Servicios de Google Play:** el teléfono móvil ha de tener instalados los Servicios de Google Play, ya que estos son usados para diversas funcionalidades de la app.

5.3.3 Interfaces de comunicación

Para conseguir una buena comunicación entre el dispositivo wearable (reloj inteligente) y el teléfono se utilizará la tecnología BlueTooth. En el caso de que el Smartwatch lo admita, la comunicación se podrá realizar también a través de redes WiFi.

5.4 Planificación Temporal

En esta sección se detallará la planificación temporal que se llevará a cabo durante la realización del proyecto. Se le asignará un número de horas estimado a cada una de las tareas en base a la experiencia del desarrollador. Se debe tener en cuenta que las horas establecidas para el Trabajo de Fin de Grado son 300, por lo que las horas estimadas para este proyecto deben aproximarse a ellas.

Cabe destacar que este proyecto se está realizando a la par que se cursa el último año de Ingeniería Multimedia y que por lo tanto sí es posible establecer una planificación temporal basada en las horas de cada tarea, pero no es posible asignarle una fecha de inicio y otra de fin, ya que el proyecto se irá realizando en la medida que las asignaturas del curso lo permitan.

TAREAS	HORAS
Investigación tecnologías o aplicaciones similares	**15**
Investigación lenguajes y entornos de desarrollo	**10**
Analizar las características de sonidos peligrosos	**15**
Grabar sonidos con el micrófono del móvil	**10**
Detectar actividad del usuario	**25**
Guardar preferencias del usuario	**10**
Detectar el sonido ambiente	**10**
Filtrar el sonido por volumen	**25**
Filtrar el sonido por frecuencia	**35**
Lanzar alerta visual	**10**
Lanzar alerta vibratoria	**5**

Conexión con el reloj inteligente	**10**
Lanzar alerta en el reloj inteligente	**10**
Diseño de la pantalla de inicio	**7**
Diseño de la pantalla Detectando	**10**
Diseño de la alerta	**5**
Diseño pantalla Configuración	**10**
Diseño alertas en el SmartWatch	**10**
Diseño del logo e iconografía	**20**
Realización de la memoria del proyecto	**100**
TOTAL	**352**

5.5 Público objetivo

Este sistema está enfocado principalmente a aquellas personas que presenten algún tipo de deficiencia auditiva. En concreto a personas que padecen dicha discapacidad y utilizan un vehículo día a día.

La aplicación también está destinada a cualquier tipo de conductor que quiera tener una información adicional mientras conduce, con el objetivo de prevenir riesgos y accidentes.

6 Diseño gráfico y desarrollo

En este apartado del documento se explicará cuál ha sido el proceso seguido para la implementación del sistema y sus principales funcionalidades así como las etapas realizadas para obtener la interfaz gráfica final.

6.1 Diseño gráfico

El diseño de la interfaz gráfica dentro de esta aplicación tiene un peso muy importante ya que el usuario debe poder reconocer tanto las alertas por sonidos peligrosos como las acciones que se están llevando a cabo con un simple vistazo. Por lo tanto, desde el primer momento el objetivo principal que se ha marcado para la interfaz gráfica es que resulte muy **intuitiva** y amigable para los distintos usuarios, consiguiendo así que el uso de la aplicación no interfiera en la conducción y que de esta manera el conductor sea capaz de centrar toda su atención en la carretera.

El diseño de la aplicación será, por lo tanto, del tipo *flat design*, ya que las características principales de esta corriente son el minimalismo, la simpleza y una alta importancia en la usabilidad. Además se combinará con las pautas y reglas del diseño introducido por Android, *Material Design*. Este se basa en objetos materiales, donde las piezas son colocadas en un espacio y con un tiempo concreto. Es un diseño donde la profundidad, las superficies, los bordes, las sombras y los colores juegan un papel principal.

Material Design quiere guiarse por las leyes de la física, donde las animaciones sean lógicas, los objetos se superpongan pero no puedan atravesarse el uno al otro y demás. **[15]**

Google pone a disposición de cualquier usuario o desarrollador una amplia guía con las buenas prácticas y las características principales que se deben seguir para respetar este tipo de diseño. Podemos encontrar esta documentación en el siguiente enlace. : https://material.google.com/.

Por otra parte, otro de los máximos marcados para el diseño gráfico de la aplicación es que sirva para cualquier tipo de dispositivo móvil, independientemente de su tamaño o forma. Por lo tanto, se debe realizar un diseño adaptativo de la aplicación. Este tipo de diseño ha de tener en cuenta que los dispositivos móviles tienen dos formas de visualizar la información, en modo portrait (vertical) o en modo landscape (horizontal), por lo que se han de diseñar interfaces distintas para cada uno de los modos.

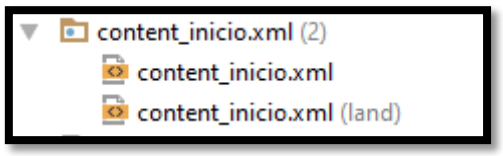


Figura 7 - Interfaces distintas para una misma actividad: portrairt y land

Fuente: Creación propia

6.1.1 Estructuración de la información

A la hora de elaborar dicho diseño, se ha tenido en cuenta la organización y disposición de los elementos en el espacio, de manera que toda la información quede uniforme en todas sus secciones.

La coherencia en la aplicación es esencial, ya que ayuda a percibir la aplicación como una sola, ganando así en robustez y usabilidad. Para ello se ha tenido especial cuidado en la colocación de los elementos que se repiten a lo largo de las distintas secciones de la app, asignándoles un lugar idéntico en cada una de ellas. Ayudamos así a la armonización de nuestro diseño.

La disposición y la apariencia que se le ha dado a la información en el diseño de la aplicación la podemos agrupar en los siguientes bloques:

6.1.2 Área Superior

1. **Fila superior:** Este espacio es la barra de notificaciones por defecto del móvil. En las nuevas versiones de Android podemos cambiar su color y hacer que esta forme parte también de nuestra aplicación, dando así una sensación mayor de unidad.

2. **Fila inferior:** Este espacio está destinado a la barra principal de la aplicación. En ella irán apareciendo los títulos de las distintas secciones, y en aquellas en las que se pueda retroceder encontraremos una flecha a su izquierda que nos llevará a la actividad de la que venimos. Su altura es fija pero su ancho es variable.

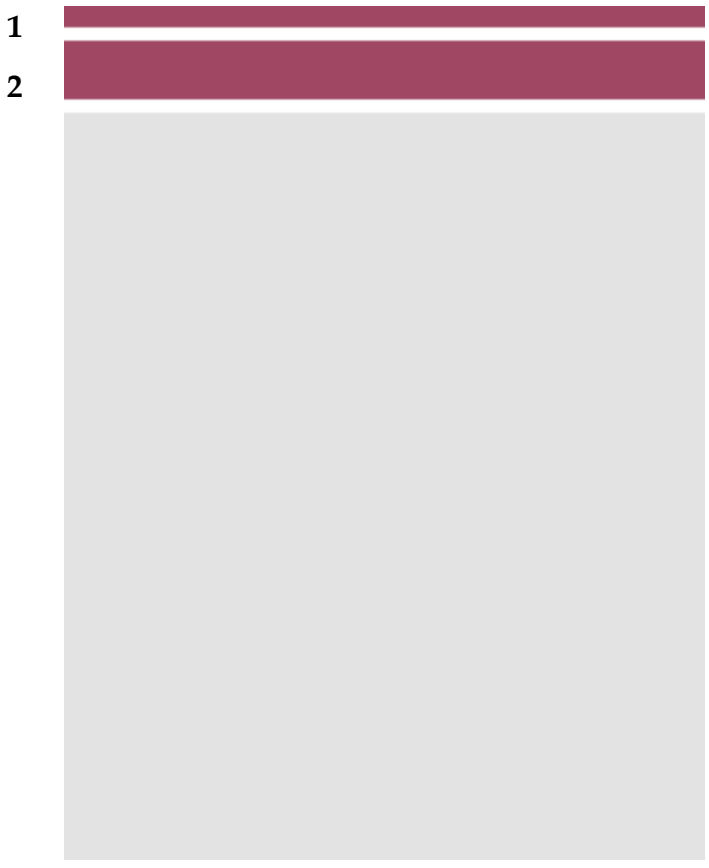


Figura 8 - Esquema 1 interfaz gráfica

Fuente: Creación propia

6.1.3 Área Informativa

1. **Área informativa:** Esta área estará reservada para todo el contenido propio de la sección de la aplicación en la que nos encontremos. Generalmente estará compuesto por un icono grande e intuitivo que indique el propósito de la sección y un texto de apoyo a dicha imagen.

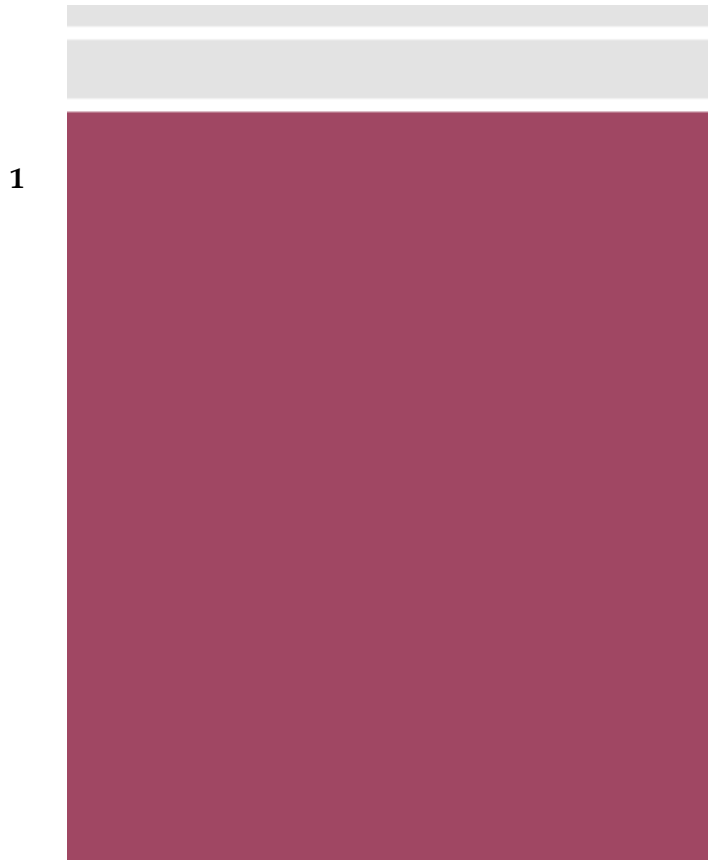


Figura 9 - Esquema 2 interfaz gráfica

Fuente: Creación Propia

6.1.4 Paleta de colores

Los colores que se han utilizado para el diseño de la aplicación son colores mayoritariamente fríos. La característica principal de estos colores es que transmiten seriedad, tranquilidad y confianza, por lo que se adaptan perfectamente al objetivo de la aplicación. Además se ha incluido algún color más cálido con la intención de llamar la atención del usuario y que este sea consciente de que el sistema se quiere comunicar con él.

La paleta de color utilizada ha sido la siguiente:

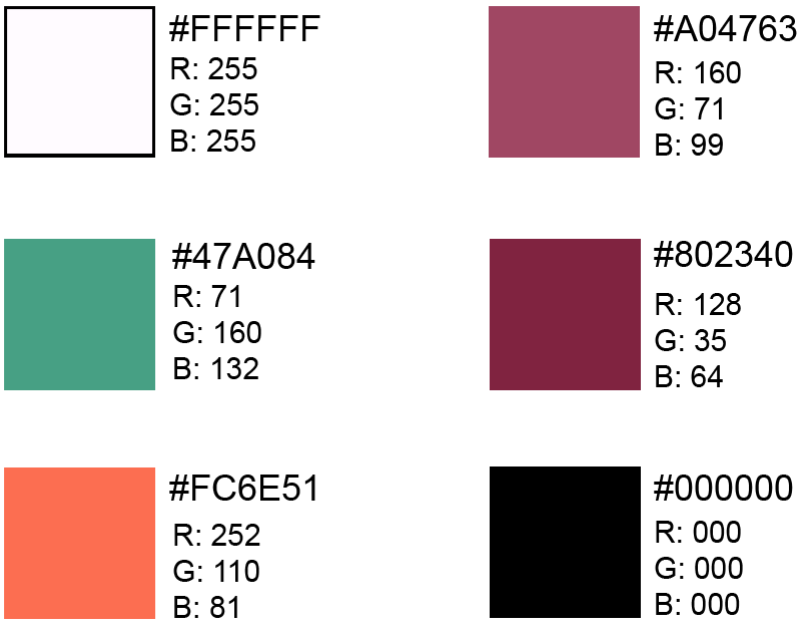


Figura 10 - Paleta de colores

Fuente: Creación propia

El color verde **#47A084** será usado para elementos complementarios en la aplicación, como por ejemplo el botón de más opciones o el interruptor para activar o desactivar alguna opción.

El color naranja **#FC6E51** será usado para algunas alertas, intentado así llamar la atención del usuario.

El color lila claro **#A04763** es el color principal de la aplicación. Este será el utilizado en los menús así como en el logotipo de la aplicación e incluso en algún texto importante.

El color lila oscuro **#802340** es el color que se le da a la barra de notificaciones del móvil para que concuerde con el resto de la aplicación. Además será usado también en algún texto de mediana importancia,

El color negro **#000000** será el utilizado en la mayoría del contenido tipográfico de la aplicación.

El color blanco **#FFFFFF** será el utilizado en los fondos así como en tipografías que se encuentren sobre un fondo de otro color diferente.

6.1.5 Tipografía

La tipografía utilizada en la aplicación es *Roboto.* Es la familia tipográfica introducida por Android con la versión 4.0 de su sistema operativo. Es del tipo *sans serif* lo cual la hace menos seria y más amigable. La familia incluye variantes delgada, liviana, regular, media, negrita, "black" y condensada.

Será la única familia tipográfica en la aplicación, pero se hará uso de todas sus variantes para que cada texto transmita la importancia y el mensaje que le corresponde.

Figura 11 - Familia tipográfica Roboto

Fuente: https://www.google.com/fonts/specimen/Roboto

6.1.6 Iconos

Como se ha comentado en la introducción a este tema, el objetivo principal del diseño gráfico de la aplicación es conseguir que el usuario sepa de un simple vistazo lo que el sistema le quiere decir. Por lo tanto, el diseño ha de ser simple e intuitivo.

Para conseguir este objetivo, se han reemplazado los textos explicativos, largos y complicados por iconos simples y representativos. De esta manera dejamos que la aplicación se comunique con el usuario de una forma rápida y eficaz, sin tener que fijar mucho la vista ni prestar demasiada atención.

En concreto, el tipo de iconos que se han elegido se denominan *flat icons* pertenecientes al tipo de diseño *flat design*. Estos iconos se caracterizan por ser simples, minimalistas y muy representativos, por lo que se adaptan a la perfección con el diseño de la aplicación.

Figura 12 - Flat icon

Fuente: http://icon-icons.com/pt/icone/fone-de-ouvido/30382

Además, debemos tener en cuenta que la aplicación también se ejecutará en dispositivos wearables, donde las pantallas son de un tamaño muy inferior al de un dispositivo móvil. Iconos de este estilo harán que la experiencia de usuario sea extremadamente satisfactoria.

6.1.7 Logo

El logo es un factor muy importante en cualquier tipo de aplicación. Con un logo original y representativo creamos marca de nuestro producto y cualquier usuario podrá reconocernos.

Como hemos comentado anteriormente, el estilo de iconos y simbología de la aplicación sigue el patrón *flat design* por lo que el logo también lo hará.

Los colores utilizados para el logo serán el lila claro **#A04763**, el blanco **#FFFFFF** y el lila oscuro **#802340** para la sombra.

Figura 13 - Logo EarDrive

Fuente: Creación propia

Como podemos observar en la siguiente imagen, el logo encaja perfectamente en el estilo utilizado por Android y el resto de aplicaciones de hoy en día.

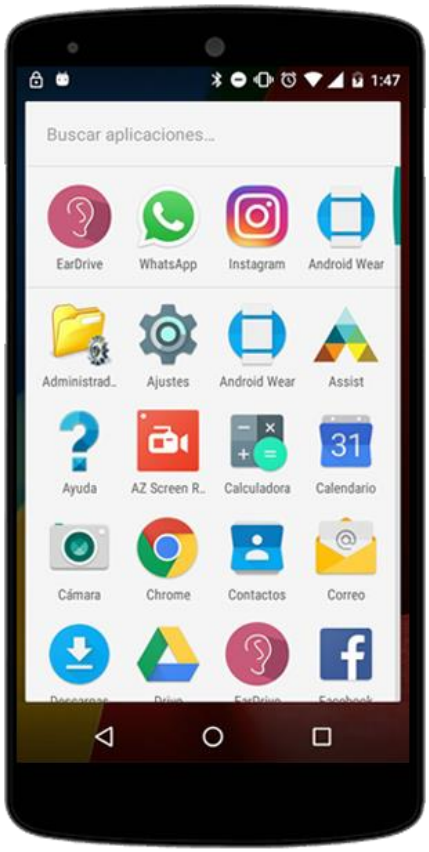


Figura 14 - Logo EarDrive en Android 6.0

Fuente: Creación propia

6.1.8 Mapa de flujo de la aplicación

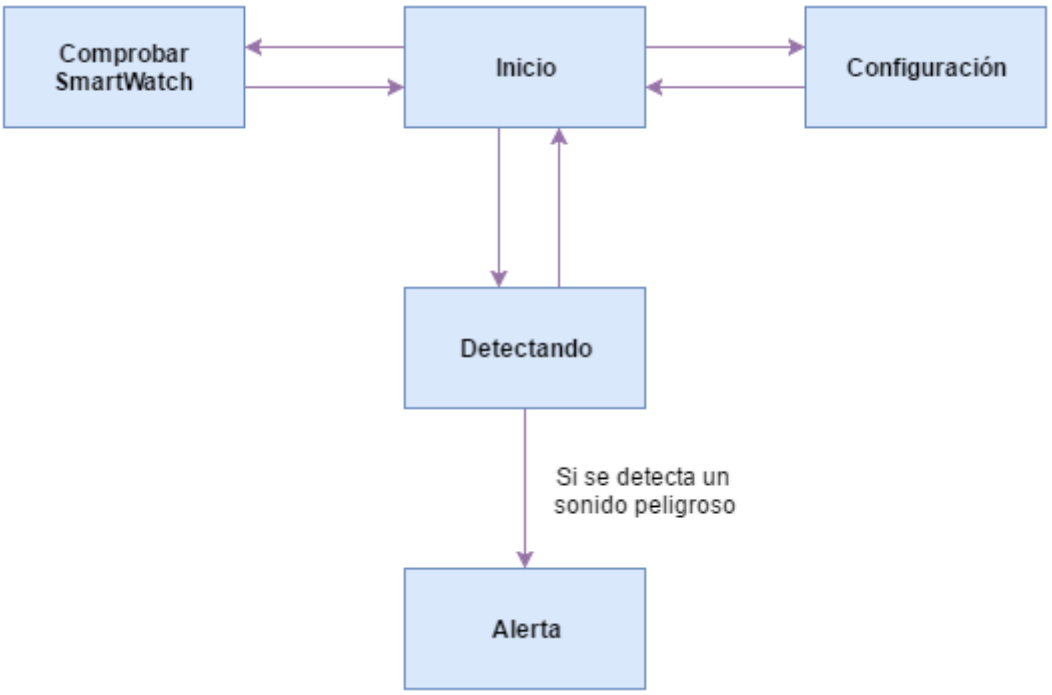
El flujo de la aplicación entre sus distintas secciones pretende ser muy simple y rápido, por lo que el número de pantallas se ha reducido al máximo.

Como podemos observar en la figura, desde la primera pantalla que le aparece al usuario (Inicio) podremos acceder a:

- **Configuración**: Pantalla donde el usuario podrá cambiar sus preferencias. Desde esta pantalla sólo podremos volver al inicio.
- **Comprobar SmartWarch**: Pantalla donde el usuario podrá comprobar si hay un reloj inteligente conectado al teléfono móvil y por lo tanto

las alertas también se visualicen en él. Desde esta pantalla sólo podremos volver al inicio.

- **Detectando:** Pantalla donde el sistema recogerá el sonido que el micrófono recibe y lo analizará. Si en este análisis se encuentra algún tipo de peligro, se lanza una pantalla de alerta. Por lo tanto, desde esta pantalla podremos volver al inicio o pasar a una alerta.
- **Alerta:** Es una ventana o pantalla emergente que se superpone a la pantalla Detectando. En ella el usuario podrá ver qué tipo de alerta ha aparecido. Esta pantalla emergente desaparece sola al cabo de 3 segundos.



6.1.9 Etapas del diseño de la interfaz

Una vez se han definido todos los aspectos importantes de la interfaz gráfica: colores, distribución de la información, tipografía y estilo a seguir, se pasa a hacer pruebas de cómo quedaría el aspecto final de la aplicación. Para esto se realizan distintos bocetos y se elige el que más se adecúa.

Versión inicial de la interfaz gráfica

Como se puede observar en las imágenes, esta versión es aún muy básica, no se han incluido colores, iconos ni diseño alternativo para adaptarse al modo *landscape* del teléfono móvil.

Es la primera etapa, con ella se pretende crear una idea general del diseño y unas ciertas pautas que se deben respetar a medida que este vaya evolucionando.

Figura 15 - Mock up pantalla inicio

Fuente: Creación propia

Figura 16-Mock up pantalla Escuchando.

Fuente: Creación propia

Figura 17 - Mock up pantalla Alerta

Fuente: Creación propia.

Segunda versión de la interfaz gráfica

En esta segunda etapa del diseño de la interfaz se han añadido bocetos alternativos para la correcta representación de los distintos tipos de alertas y para el modo *landscape* de los teléfonos móviles.

Como podemos observar en las imágenes, todos estos bocetos siguen ya el mismo patrón de diseño que se aprobó en la primera etapa.

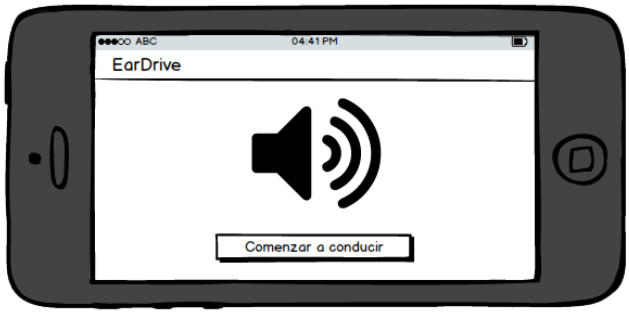


Figura 18 - Mock up landscape pantalla inicio

Fuente: Creación propia

La siguiente imagen se correspondería a un tipo de alerta específico.

Figura 19 - Mock up landscape alerta específica

Fuente: Creación propia

Se puede comprobar en los bocetos que además de haber adaptado su diseño al modo *landscape* también se han eliminado elementos de la interfaz que sí aparecían en la primera etapa del diseño, en concreto elementos de la barra superior de la aplicación.

- El logo: se elimina el logo en la barra superior para así dar impresión de un diseño más limpio y simple.
- El menú: se elimina también el menú de la barra ya que no se le va a dar uso en la aplicación.

Tercera versión de la interfaz gráfica

En esta tercera etapa del diseño se añaden los colores definidos en la paleta y se incorpora el logo y algunos iconos importantes.

Los diseños definidos en esta etapa han de cuidar más los detalles que en las etapas anteriores, ya que la interfaz gráfica final de la aplicación será la implementación de estos mismos diseños en la plataforma escogida (Android).

Figura 21 - Mock up pantalla inicio con colore.

Fuente: Creación propia

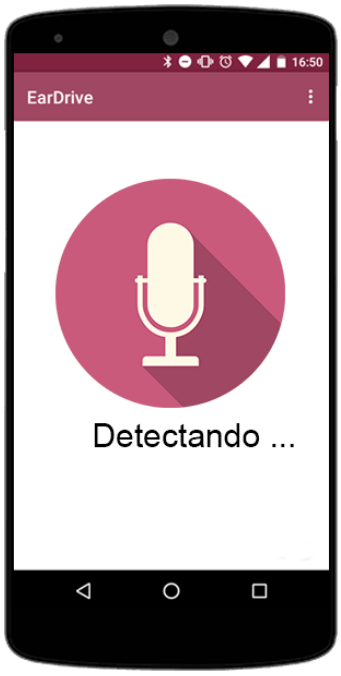


Figura 20 - Mock up pantalla detectando con colores e icono

Fuente: Creación propia.

Versión final de la interfaz gráfica

Este diseño es el definitivo de la aplicación. Es el resultado de implementar todos los bocetos enseñados anteriormente en Android. Como se puede observar en las

imágenes, en esta última etapa se han hecho también pequeños retoques que mejorarán la experiencia del usuario.

- Se ha cambiado el botón de la pantalla de inicio por un texto simple e intuitivo.
- Se ha añadido una animación en la pantalla de detección que solamente estará activa cuando el móvil está grabando el sonido ambiente.
- Se ha añadido en la pantalla de inicio un botón flotante que nos abrirá las opciones adicionales de la aplicación.

Figura 23 - Pantalla inicio

Fuente: Creación propia.

Figura 22 - Pantalla detectando

Fuente: Creación propia.

6.1.10 Control de versiones

Se debe tener en cuenta a la hora de realizar el diseño que no todas las versiones del sistema operativo Android soportan los mismos elementos en las interfaces. Las versiones más nuevas han incluido elementos que no son compatibles con versiones anteriores, por lo que se debe realizar un diseño alternativo para todos aquellos elementos que no sean admitidos por todas las versiones. Un claro ejemplo de esto

es el elemento Switch que se utiliza para activar las preferencias, este elemento únicamente es soportado por las versiones de la API de Android superiores a la 14, por lo que tendremos que diseñar una interfaz alternativa en esta actividad.

En el caso de que estos diseños alternativos no se realicen, perderíamos accesibilidad en la aplicación y por lo tanto perderíamos también a posibles usuarios activos.

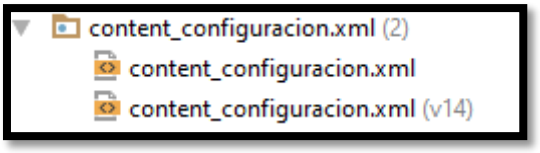


Figura 24 - Diseños alternativos para diferentes versiones de Android

Fuente: Creación propia

6.2 Diccionario de sonidos

Este apartado no incluye desarrollo ni implementación de la aplicación, pero es una parte fundamental para el correcto funcionamiento de ésta. Se analizarán los sonidos que pueden representar un peligro en la conducción y se determinarán cuáles son sus principales características para que posteriormente el sistema las pueda comparar con el sonido que le entra al dispositivo móvil por el micrófono y lanzar las alertas al usuario si existen coincidencias.

Para poder obtener las características principales de aquellos sonidos que constituyen una alerta auditiva analizaremos algunas de las cualidades principales del sonido:

- **Intensidad:** Determina si un sonido es fuerte o débil y depende directamente de la amplitud. Cuanto mayor sea la amplitud más fuerte será el sonido.

- **Tono:** Determina si un sonido es agudo (tono alto) o es grave (tono bajo). Depende directamente de la frecuencia, cuanto mayor sea la frecuencia más alto será el tono.

En esta versión primitiva del sistema los sonidos serán filtrados por volumen y por frecuencia, tal y como se explica en el punto 9.3 Desarrollo. Por lo tanto, serán estas características las que analizaremos de los sonidos que puedan representar un peligro en la conducción.

Uno de los sonidos básicos y más comunes que se encuentran los conductores en las carreteras es el claxon. Éste es el claro ejemplo de una alerta auditiva, ya que siempre indica un peligro o una advertencia por parte de otro conductor, por lo tanto este será el primer sonido que analizaremos y que contendrá nuestro diccionario.

Lo que debemos hacer es analizar las frecuencias de dicho sonido y determinar cuál es su frecuencia fundamental, cuál es su frecuencia con más energía y cuáles son sus parciales más representativos. Todos estos datos son los que utilizará el algoritmo del sistema para poder filtrar los sonidos por frecuencia.

Toda esta información la encontraremos analizando el espectrograma del sonido. De esta forma, si estudiamos dichas características de un sonido estándar de un claxon obtendremos que la frecuencia fundamental se encuentra en 400 Hz, que la frecuencia que más energía aporta a la señal es 1200 Hz, es decir, el tercer parcial, y que los parciales más representativos son el segundo (800 Hz) y el tercero. Esto lo podemos observar en la siguiente imagen obtenida con el programa Sonic Visualiser.

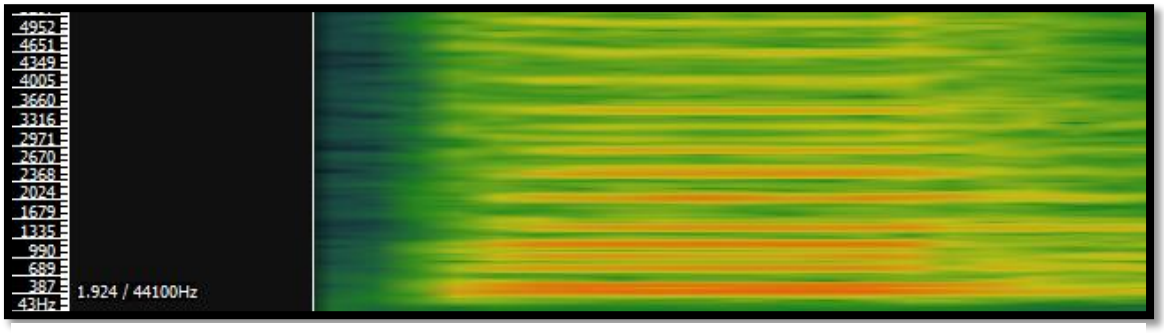


Figura 25 - Espectrograma de un claxon

Fuente: Creación propia

Serán estos valores los que utilizará el sistema para determinar si el sonido que está escuchando el teléfono móvil es un claxon o es algo diferente. Cabe destacar que en esta primera versión del sistema este tipo de filtrado sólo se utiliza para alertas auditivas provocadas por un claxon. Pero los pasos que se han seguido para esta detección son los que se deben seguir para detectar cualquier otro tipo de sonido complejo que pueda ser considerado como peligroso para un conductor:

- Sirenas
- Frenazos
- Gritos de peatones
- Choques. Estos sucesos suelen tener una intensidad de sonido alta por lo que con el filtrado por volumen se podría detectar.

Los valores obtenidos en el análisis de los sonidos listados arriba se incluirían en el diccionario de sonidos y el sistema los tendría en cuenta a la hora de filtrar los ruidos que recibe.

Además, en la conducción no todos los sonidos peligrosos nos resultan familiares, por lo tanto el sistema no será capaz de reconocerlos. Se debe entonces encontrar alguna propiedad en común que le ayude a diferenciarlos. Algo muy característico de los sonidos que representan una alerta en la conducción es su volumen, su intensidad. Cualquier ruido o sonido que suene más fuerte que el volumen del ruido ambiente llamará la atención del conductor y lo pondrá en alerta. De esta forma, el sistema debe ser capaz de discriminar el ruido ambiente y detectar aquellos sonidos que

lo sobrepasen de una manera amplia. El algoritmo considerará que dichos sonidos representan una alerta auditiva para el conductor. En el punto 9.3.5.1 Filtrado por Volumen se explica el proceso de filtrado que sigue el algoritmo.

6.3 Implementación

En este apartado se explicará el desarrollo de la aplicación y sus funcionalidades más importantes.

6.3.1 Preferencias de usuario

Una funcionalidad importante de la aplicación es el poder guardar las preferencias del usuario. Se podrá acceder y personalizar estas preferencias en la pantalla de 'Configuración', allí nos encontraremos con opciones que se podrán activar, desactivar o cambiar sus valores dependiendo de los intereses del usuario.

- **Alertas vibratorias:** Cuando se detecte algún peligro y el sistema determine que se debe avisar al usuario lo hará con una alerta visual siempre y podrá estar acompañada de una alerta vibratoria si el usuario así lo ha configurado.
- **Conexión con el SmartWatch:** Cuando el sistema detecte algún peligro y determine que se debe avisar al usuario, lo hará con una alerta en el dispositivo móvil siempre y podrá enviar esta alerta a un reloj inteligente si el usuario así lo ha configurado. Para que esta funcionalidad se complete correctamente debe haber un reloj emparejado con el móvil.
- **Parar la detección automáticamente según la actividad del usuario:** Si el usuario activa esta opción el sistema se encargará de detectar cuál es la actividad del usuario en cada momento. Cuando se detecte que el usuario se encuentra quieto fuera de un vehículo, el sistema dejará de grabar y analizar el sonido ambiente.
- **Valor del umbral:** Uno de los tipos de filtrado que implementa el sistema es por volumen. Por lo tanto, en la pantalla de 'Configuración' el usuario podrá modificar el umbral que desea que tenga dicho filtro. Cuánto más

alto el umbral más alto debe sonar el sonido para que represente una alerta auditiva.

- **Peligros a detectar:** El usuario podrá configurar cuales son los sonidos que desea que el sistema reconozca y le avise cuando los escuche.

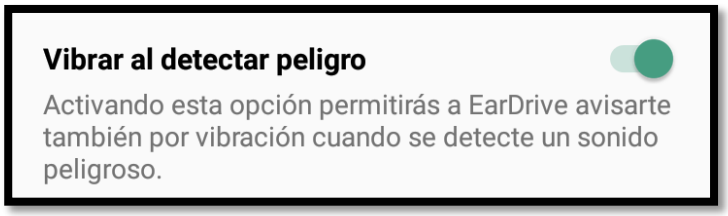


Figura 26 - Ejemplo preferencias de usuario

Fuente: Creación propia

Para poder implementar está funcionalidad se ha utilizado la API Shared Preferences. Dado que los datos que vamos a almacenar son una colección muy pequeña y podrán ser del formato clave - valor, esta API nos proporciona todo lo necesario, evitando así el uso de una base de datos y que la aplicación aumente su complejidad y tamaño. La información almacenada con Shared Preferences permanecerá intacta aunque cerremos la aplicación o reiniciemos el dispositivo móvil.

Se utilizará un objeto Shared Preferences que apuntará a un archivo que contiene pares de clave – valor y además nos proporcionará métodos rápidos y sencillos para poder escribir y leer en él. Podremos almacenar valores de cualquier tipo primitivo (booleans, floats, ints, longs, y strings) pero en el caso de la aplicación utilizaremos booleanos para describir si las opciones están activa o inactivas y valores enteros para almacenar el valor del margen.

6.3.2 Detección de la actividad del usuario

Como se ha explicado en al apartado de Preferencias de usuario, esta funcionalidad es optativa. Es decir, podrá estar activa o inactiva, dependiendo de lo que el usuario desee y configure.

El sistema se encargará de detectar cual es la actividad que está realizando el usuario en cada momento. Si la actividad detectada es 'Quieto', el sistema detendrá la grabación y el análisis de los sonidos externos, evitando así que el consumo de batería del usuario se dispare si este se olvida de parar la aplicación cuando abandona el vehículo. Se pretende con esta funcionalidad mejorar la experiencia de usuario y la usabilidad de la aplicación, intentando corregir y prevenir los errores humanos.

Para reconocer la actividad que el usuario está realizando utilizaremos la API Google Location Services perteneciente a los Servicios de Google Play. Los tipos de actividades que podemos detectar son los siguientes:

- En un vehículo
- En bicicleta
- Caminando
- Corriendo
- Quieto
- Actividad desconocida (valor que devuelve la API cuando no es capaz de determinar la actividad)

La API no nos devolverá una actividad en concreto y real, si no que devolverá el porcentaje de probabilidad de que cada una de esas actividades se esté realizando, por lo tanto es responsabilidad del desarrollador de la aplicación el interpretar esos datos y actuar en consecuencia.

En el caso de este proyecto, se ha tomado como valor mínimo un 80%, es decir, solamente se parará de detectar sonidos si el porcentaje de confianza de la actividad 'Quieto' es mayor a un 80%. Siempre y cuando esté esta opción habilitada.

Para que el reconocimiento de la actividad se adapte de manera correcta a lo que el usuario está haciendo, debemos realizar llamadas de forma periódica a la API de Google Services e ir cambiando y comprobando los valores. Para esto se define un tiempo de actualización, el cual debe estar cuidadosamente seleccionado ya que tiempos muy largos perjudicarían al funcionamiento de la aplicación, mientras que

tiempos muy cortos incrementarían el consumo de batería. Además, debemos tener en cuenta que las respuestas de la API se pueden retrasar bastante y no respetar el tiempo de actualización si el servicio de detección necesita más muestras para poder hacer una predicción más acertada.

```
06-18 23:29:47.757 21629-21629/? I/MainActivity: Activity: Unknown activity, Confidence: 62%
                                                  Activity: Still, Confidence: 19%
                                                  Activity: In a vehicle, Confidence: 17%
                                                  Activity: On a bicycle, Confidence: 2%
```

Figura 27 - Actividades reconocidas por el sistema

Fuente: Creación propia

6.3.3 Conexión con el reloj inteligente

Con esta funcionalidad el sistema será capaz de comunicar las alertas detectadas por sonidos peligrosos al dispositivo "wearable" que se encuentre emparejado con el teléfono móvil. En el caso de que hubiese más de un reloj emparejado, se lo enviaría a todos.

Es una funcionalidad básica de la aplicación. Se pretende con ella que el conductor del vehículo no tenga que estar pendiente de la pantalla de su dispositivo móvil y pueda centrar toda su atención en la conducción.

Las alertas en el reloj serán visuales pero también vibratorias, consiguiendo así que el usuario se entere en todo momento de lo que está detectando el sistema y no desvíe la mirada de la carretera. Para esto se establecerá un patrón de vibración largo y personalizado, con el objetivo de que ésta sea la única aplicación con dicha vibración y por lo tanto fácilmente reconocible.

Para conseguir que esta comunicación entre dispositivo móvil y reloj inteligente sea posible se utilizará la API para Wearables (WearableAPI). Esta API incluye tres nuevas APIs para ayudar a la comunicación y hacer este proceso más simple: Message API, Node API y DataLayer API. Cada una de ellas tiene su propio objetivo; Message

API está diseñada para 'lanzar y olvidar' cualquier tipo de mensaje, DataLayer API soporta la sincronización de información entre el reloj y el móvil, mientras que Node API controla eventos relacionados con los dispositivos conectados y emparejados. En la aplicación se hará uso de MessageAPI para enviar las alertas al reloj, y de NodeAPI para encontrar que dispositivos hay conectados.

Por otra parte, se debe tener en cuenta que la pantalla de los dispositivos "wearables" es de dimensiones pequeñas y que por lo tanto la alerta debe ocupar toda la pantalla. Las notificaciones en los relojes inteligentes se muestran en un espacio pequeño de la pantalla, no llegan a ocupar su totalidad, por lo que se ha descartado esta posibilidad. De esta manera, concluimos en que la mejor opción es crear una aplicación también para wearables y que esta tenga su propia actividad y con su propio diseño, pudiendo así modificarlo y adaptarlo a nuestro parecer.

La aplicación para wearables ha de estar siempre escuchando a los mensajes que le pueden llegar desde el móvil, por lo que se debe ejecutar en segundo plano y hacer aparecer una alerta cuando se le indique. Para conseguir esto se ha implementado un Listerner (*WearableListenerService)* de la aplicación. Este servicio estará siempre atento a los mensajes que se reciben desde el dispositivo móvil, y cuando llegue uno que esté relacionado con la aplicación se lanzará la actividad con la alerta correspondiente.

Figura 28 - Esquema de comunicación entre móvil y reloj

Fuente: Creación propia

6.3.4 Captación del sonido

Funcionalidad básica de la aplicación. Con ella el dispositivo móvil será capaz de escuchar todo lo que sucede a su alrededor y analizar los sonidos para determinar si existen peligros para el usuario.

Para poder capturar el sonido desde el micrófono del dispositivo utilizaremos la clase AudioRecord que nos proporciona Android. Crearemos un objeto de dicha clase al cual le indicaremos los parámetros con los que queremos que el audio sea capturado y grabado. Debemos tener en cuenta que todo lo que recoge el micrófono ha de ser analizado posteriormente para determinar si existe algún peligro, por lo que no sirve simplemente 'grabar y olvidar' si no que tenemos que seleccionar muy cuidadosamente que características necesita el audio para poder ser analizado correctamente y capturarlo con los mismos parámetros.

Los parámetros que debemos tener en cuenta a la hora de capturar el sonido serán los siguientes:

- **Audio Source:** será la entrada de audio desde la que se capturará el sonido. Puede ser un fichero externo pero en el caso de la aplicación se utilizará el micrófono del dispositivo, ya que se pretende que el análisis de la señal sea en tiempo real.
- **Frecuencia de muestreo:** La frecuencia de muestreo es el número de muestras por unidad de tiempo que se toman de una señal continua para producir una señal discreta, durante el proceso necesario para convertirla de analógica en digital. Como todas las frecuencias, generalmente se expresa en hercios (Hz) o múltiplos suyos, como el kilohercio (kHz) **[13].** Para la aplicación necesitamos una buena calidad del sonido para poder determinar qué es lo que está sonando en cada momento, por lo que la frecuencia de muestreo que se utilizará será de 44.100 Hz, es la más utilizada y la normal para grabar con calidad de CD.
- **Número de canales:** Hace referencia al número de pistas que se utilizarán para grabar el sonido que llega al micrófono. Dos valores son admitidos por la clase AudioRecord: **Mono** y **Estéreo**. Si grabamos nuestro sonido en estéreo estaremos utilizando dos canales, uno izquierdo y otro derecho, por lo que el sonido obtenido será mucho más realista. Por otra parte, si grabamos en Mono sólo se utilizará un canal, por lo que se perderá calidad de audio obtenido.

  Para poder seleccionar el número de canales más adecuado debemos tener en cuenta también que nuestro sistema ha de analizar el sonido en tiempo real, por lo que si utilizamos 2 canales se tendrá que analizar el doble de muestras, pudiendo esto perjudicar al tiempo de respuesta de la aplicación. De esta manera, concluimos en que el mejor modo de grabación es **mono**, ya que aunque se pierda realismo, es suficiente para poder determinar que sonido se está escuchando y además mejoraremos el tiempo de análisis de las muestras obtenidas. Por otra parte, la documentación de Android nos aconseja utilizar un canal mono ya que está garantizado que funcione en todos los dispositivos. **[17]**

- **Formato de audio:** Hace referencia a la codificación utilizada para describir los bits del audio obtenido, los cuales pueden ser valores enteros PCM o audio comprimido como por ejemplo AC3 o DTS. En la aplicación que se está desarrollando se necesitan los valores del audio en 'bruto' por lo tanto no debemos utilizar formatos de audio comprimido. Se utilizará la codificación **PCM**.

  Los valores que se codificarán en PCM representan la magnitud de la señal sonora analógica en el momento que se toma dicha muestra. Por lo tanto, un conjunto o flujo de valores PCM será la representación digital de una señal analógica, donde la magnitud de la onda es tomada en intervalos uniformes (Frecuencia de muestreo) y cada muestra puede tomar un conjunto de valores finito. Este número de posibles valores que puede tomar cada una de las muestras viene definido por la profundidad en bits.

  La clase Audio Record nos permite 3 tipos de valores en el formato de Audio:

  1. **PMC 8 bits:** La muestra de audio será un entero de 8 bits sin signo. Es decir, podrá tomar valores entre 0 y 255 (2^8 -1).
  2. **PCM 16 bits:** La muestra de audio será un entero de 16 bits con signo, por lo tanto podrá tomar valores entre -32768 y 32767].
  3. **PCM FLOAT:** La muestra será un número en coma flotante de 32 bits.

  Como se ha comentado anteriormente, el formato de audio utilizado para la codificación de la señal será PCM y se completará con una profundidad de 16 bits. Se ha descartado la opción de 8 bits ya que reduciría mucho la calidad del audio y además no es soportada por todos los dispositivos. Lo mismo sucede con la opción de PCM FLOAT, es un tipo de codificación introducido a partir de la versión Lollipop de Android, por lo que muchos dispositivos no la soportarían.

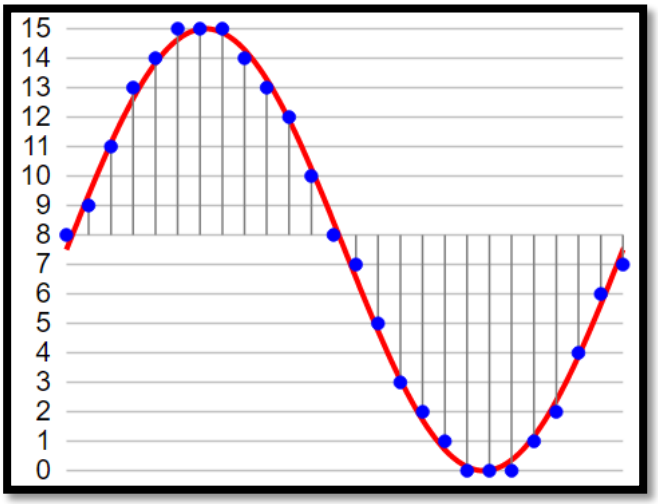


Figura 29 - Muestreo y cuantificación de una onda en PCM 4 bits

Fuente:

https://es.wikipedia.org/wiki/Modulaci%C3%B3n_por_impulsos_codificados#/media/File:Pcm.svg

- **Tamaño del buffer:** La clase AudioRecord irá almacenando las muestras obtenidas de la señal analógica en un buffer, por lo que antes de comenzar a grabar el sonido debemos definir su tamaño. Este tamaño definirá cuanto tiempo y muestras puede grabar el AudioRecord antes de comenzar a sobrescribir valores que no han sido leídos aún. Como nuestra aplicación trabaja en tiempo real con las muestras, no nos interesa que el tamaño del buffer sea muy grande, ya que esto podría retrasar el tiempo de respuesta de la aplicación. Por lo tanto, crearemos el buffer con el tamaño mínimo permitido por el dispositivo según el formato del audio, el número de canales y la frecuencia de muestro:

```
bufferSize = AudioRecord
        .getMinBufferSize(sampleRate,
AudioFormat.CHANNEL_IN_MONO,
                AudioFormat.ENCODING_PCM_16BIT);
```

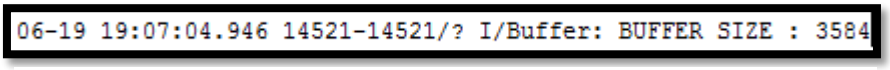


Figura 30 - Tamaño del buffer de audio

Fuente: Creación propia

Una vez se han definido cuales son los parámetros con los que queremos capturar el sonido podremos empezar la grabación. La clase irá almacenando las muestras y se podrá acceder a ellas a través del método *read* que AudioRecord nos proporciona. Este método será muy útil para poder analizar el sonido y determinar si existe algún peligro.

6.3.5 Análisis de la señal

Es la funcionalidad más importante del sistema. Con ella trataremos todo el audio grabado por el micrófono del dispositivo y se determinará si existe algún peligro.

Como se ha explicado en el apartado 9.2.4 Captación de sonido, las señales sonoras que lleguen al teléfono móvil serán muestreadas y almacenadas en un buffer de audio. En este buffer encontraremos las distintas magnitudes que tiene la señal analógica del sonido en codificación PCM 16 bits.

Si observamos la figura 28 podremos entender mejor el mecanismo y los valores con los que nos encontraremos. En el buffer de audio tendremos todos los valores que representan los puntos azules en la señal. En el caso de la figura solo pueden tomar valores entre 0 y 15, ya que la profundidad de bits es de 4, pero nuestro sistema será de 16 bits con signo, por lo que en el buffer podrá haber valores entre -32768 y 32767. Concluimos entonces que los valores que almacena el buffer representan la **amplitud** de la señal analógica.

Una vez tenemos los valores de la amplitud en el buffer podremos procesarlos y filtrarlos con el objetivo de encontrar las características propias de un sonido peligroso y así poder lanzar la alerta. En la aplicación se han realizado dos tipos de filtrado: por volumen y por frecuencia.

6.3.5.1 Filtrado por volumen

El objetivo de este tipo de filtrado es detectar aquellos sonidos que tengan un volumen muy alto y que por lo tanto puedan significar un peligro para el conductor, como por ejemplo un choque entre dos vehículos.

La psicoacústica del sonido nos explica que la sonoridad es la medida subjetiva con la que el oído humano percibe la intensidad de un sonido. Está fundamentalmente relacionada con la potencia o intensidad sonora y ésta a su vez viene determinada por el cuadrado de la amplitud de la onda.

Lo que mide la potencia de una onda que llega a nuestros oídos es la intensidad o nivel de presión sonora (NPS o simplemente nivel) que es función de la amplitud de la onda de presión medida en newtons por metro cuadrado (N/m^2) o pascales (Pa). **[18]**

Por lo tanto, con los valores de amplitud que tenemos en buffer del audio podremos calcular la sonoridad de las señales que llegan al micrófono y determinar si un sonido es suficientemente fuerte como para representar un peligro. Pero la sonoridad en sistemas digitales se calcula utilizando decibelios full scale (DBFs) y no decibelios SPL, la diferencia principal es que en lugar de utilizar el umbral auditivo como valor de referencia se utilizará la amplitud máxima del sistema. Por lo tanto, la fórmula a utilizar será la siguiente:

$$DBFs = 20 * log_{10}(\frac{A}{A_{max}})$$

A: *amplitud de la señal, cada uno de los valores del buffer de audio.*

A_{max}***:*** *amplitud máxima del sistema.*

En el caso de nuestra aplicación $\mathbf{A_{max} = 32768}$ dado que el sistema tiene una profundidad de 16 bits, por lo tanto $\mathbf{A_{max} = 2^{15}}$**.** De esta manera, podemos calcular la sonoridad en el sistema con la siguiente fórmula:

$$DBFs = 20 * log_{10}(\frac{A}{2^{15}})$$

La fórmula muestra como a mayor amplitud mayor sonoridad, y que por lo tanto la intensidad del sonido será más fuerte, es decir, tendrá un volumen más alto.

Una vez que se ha calculado la sonoridad, se establece un **umbral máximo** y todo aquel sonido que sobrepase dicho umbral será considerado como un sonido peligroso para el conductor, y por lo tanto se lanzará una alerta para el usuario.

Se debe tener en cuenta que dentro de un vehículo pueden existir sonidos fuertes que no representen ningún riesgo, como por ejemplo música con un volumen alto, una conversación acalorada o el simple ruido del resto de vehículos de la carretera. Para evitar que las alarmas salten con todos estos sonidos, la aplicación graba el sonido ambiente durante 1 segundo y medio antes de comenzar a detectar peligros. De esta forma, el sistema es capaz de conocer cuál es el volumen real del interior del vehículo y comenzar a discriminar alertas sonoras a partir de esta base. De esta forma, el umbral máximo que se establece para filtrar los sonidos fuertes no será estático, sino que será adaptativo. Se calculará cuál es el sonido ambiente del vehículo y se establecerá un umbral en relación a él.

Además, si el usuario determina que el umbral calculado automáticamente por el sistema es erróneo, ya sea porque es demasiado sensible o está demasiado alto y no es capaz de reconocer sonidos peligrosos, podrá modificarlo y ajustar dicha sensibilidad a su gusto.

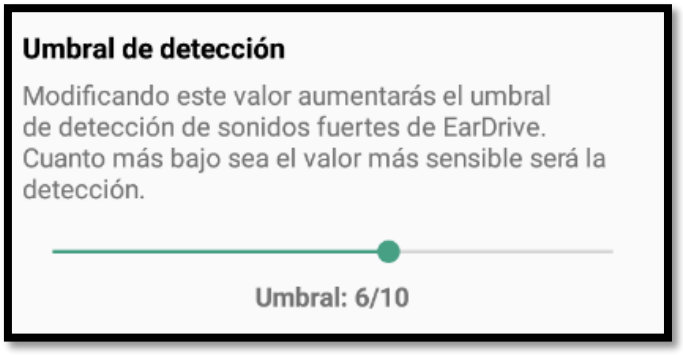


Figura 31 - Slider con el que el usuario podrá modificar el umbral de detección de sonidos fuertes

Fuente: Creación propia

Para calcular el sonido ambiente el sistema tomará muestras durante 1 segundo y medio y sacará la media de los valores obtenidos. Esto se debe a que una canción o una conversación no tienen en todo momento el mismo volumen y por lo tanto no tienen la misma amplitud, por lo que lo más adecuado es determinar cuál es el valor medio de dichos valores.

El umbral será la suma de este sonido ambiente y un margen para evitar falsos positivos cuando el sonido sobrepase el sonido ambiente pero no lo suficiente como para representar una alerta.

El umbral se calculará de la siguiente manera:

$$umbral = sonido\ ambiente + margen$$

El margen serán los DBFs que el sistema deja por encima del valor del sonido ambiente hasta que el sonido represente un riesgo. Es decir, si por ejemplo, el sonido ambiente tiene un valor de -15 BDFs y el margen es de 5 DBFs, sólo los sonidos con un valor superior a -15 + 5 = -10 DBFs serán tomados como un peligro. Este margen será el que podrá modificar el usuario al cambiar los valores del slider de la figura 31.

6.3.5.2 Filtrado por frecuencia

Para poder discriminar distintos tipos de sonidos y determinar peligros en base al sonido que suena y no a su volumen hemos utilizado la frecuencia.

Hasta ahora se estaba analizando el sonido en el dominio temporal, pero para poder discriminar los sonidos por frecuencias debemos trabajar en el dominio frecuencial. Para esto utilizaremos el algoritmo de la Transformada Discreta de Fourier (DFT), **la Transformada Rápida de Fourier (FFT).** Con ella conseguiremos el espectro de la señal y seremos capaces de determinar cuál es la frecuencia fundamental del sonido y que frecuencia es la que tiene mayor energía de toda la señal sonora que recibimos.

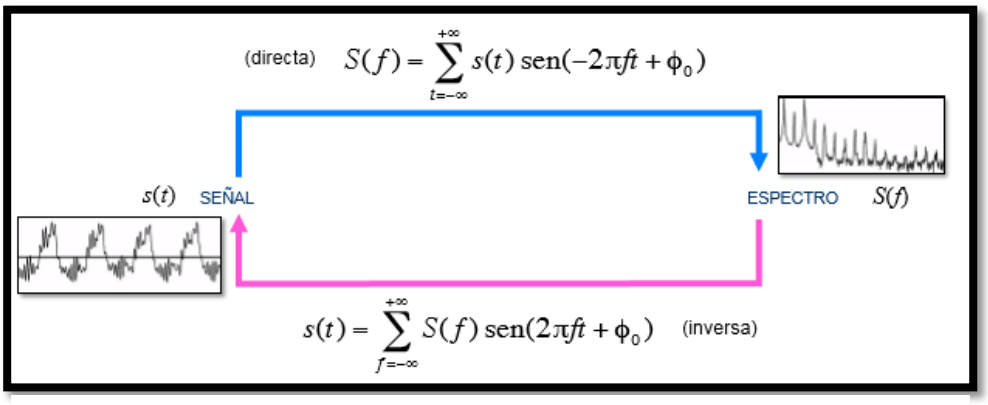


Figura 32 - Paso del dominio temporal al frecuencial con la DFT y su inversa

Fuente: Diapositivas de la asignatura Sonido y Música por Computador

Sabemos que un determinado sonido puede representarse matemáticamente como una combinación lineal de funciones sinusoidales. Cada sinusoide aporta una determinada frecuencia a ese sonido y con una determinada energía. Por lo que para poder reconocer un sonido en concreto debemos saber cuál es la frecuencia de la función sinusoide que más energía aporta a la señal original. En definitiva, debemos encontrar que frecuencia tiene la mayor amplitud, ya que ésta será la más característica del sonido.

Para esto aplicaremos la **FFT** a nuestro buffer de audio, y obtendremos así el espectro de la señal que hemos muestreado. Debemos buscar entonces dentro de este espectro cuál es la frecuencia que más energía aporta a la señal original.

Cuando le aplicamos la FFT al buffer de audio obtendremos un array del mismo tamaño y sus valores serán las diferentes magnitudes de las frecuencias de la señal, es decir, estarán representadas las magnitudes de frecuencias entre 0 y la frecuencia de muestreo (fs), que en el caso de la aplicación es 44.100 Hz.

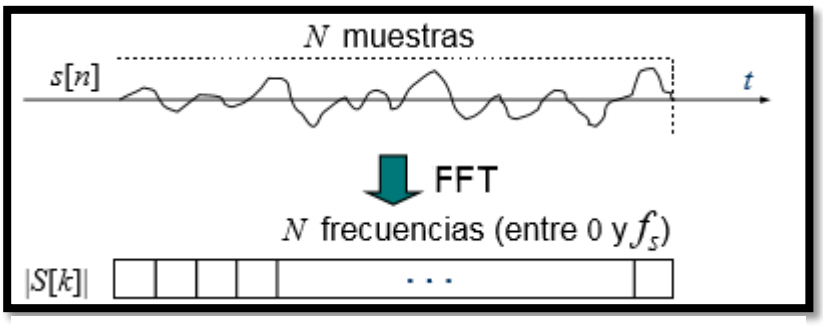


Figura 33 - Esquema de la FFT donde s[n] sería el buffer de audio y s[k] sería la salida de la FFT

Fuente: Diapositivas de Sonido y Música por Computador

Por lo tanto, debemos recorrer este array buscando la mayor amplitud y cuando la encontremos calcularemos la frecuencia en la que se encuentra. La frecuencia la obtendremos de la siguiente manera:

$$frecuencia = \ posicion * \frac{frecuencia\ de\ muestreo}{numero\ de\ muestras}$$

Donde *posición* representa la posición en la que se encuentra la amplitud dentro del array y *número de muestras* representa el número de posiciones que tiene el array.

Este array sólo se recorrerá hasta la mitad de sus posiciones, *Número de muestras*/2, ya que según la teoría de Fourier y la propiedad de simetría del espectro, las frecuencias en la segunda mitad de este array serán reflejos de las de la primera mitad. Por lo tanto las únicas frecuencias útiles serán las que se encuentren entre 0 y *número de muestras*/2, y por lo tanto entre 0 y *fs*/2. Este es un dato que debemos tener muy en cuenta, ya que mejorará el tiempo de respuesta del algoritmo y por lo tanto la experiencia de usuario de la aplicación. **[19]**

Una vez que conocemos cuál es la frecuencia que aporta más energía a la señal original podremos compararla con nuestro diccionario de sonidos y comprobar si coincide con alguno de ellos.

Para mejorar el filtrado podemos comprobar también que existe energía en el resto de parciales del espectro de la señal y no sólo en el parcial con mayor amplitud. Esto significa que si el espectro de nuestra señal fuese como el de la figura 34, debemos comprobar también que existe energía en los parciales 2 y 4 por ejemplo. De esta manera, realizando todas estas comprobaciones filtraremos mejor la señal y obtendremos menos falsos positivos.

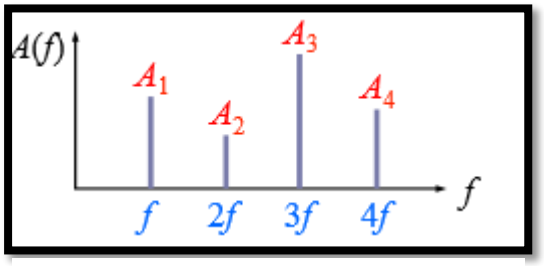


Figura 34 - Espectro en amplitud de una señal sonora

Fuente: Diapositivas de Sonido y Música por Computador

Se debe tener en cuenta que los sonidos en la realidad no son estáticos, son cambiantes y por lo tanto lo más práctico es analizarlos en plazos cortos de tiempo. Para esto utilizaremos una función ventana, la cual tendrá un tamaño fijo de muestras (N) y anulará el resto de valores de la señal. De esta forma, sólo se aplicará la FFT a aquellos valores que se encuentren comprendidos dentro de la ventana. En la aplicación que se está desarrollando se ha utilizado una ventana Hamming con un tamaño de 1024 muestras.

Por lo tanto, para obtener el espectro de la señal y aplicar los pasos que se han descrito anteriormente para conseguir la frecuencia con mayor amplitud debemos multiplicar la señal por la función ventana y al resultado de esta multiplicación aplicarle la FFT.

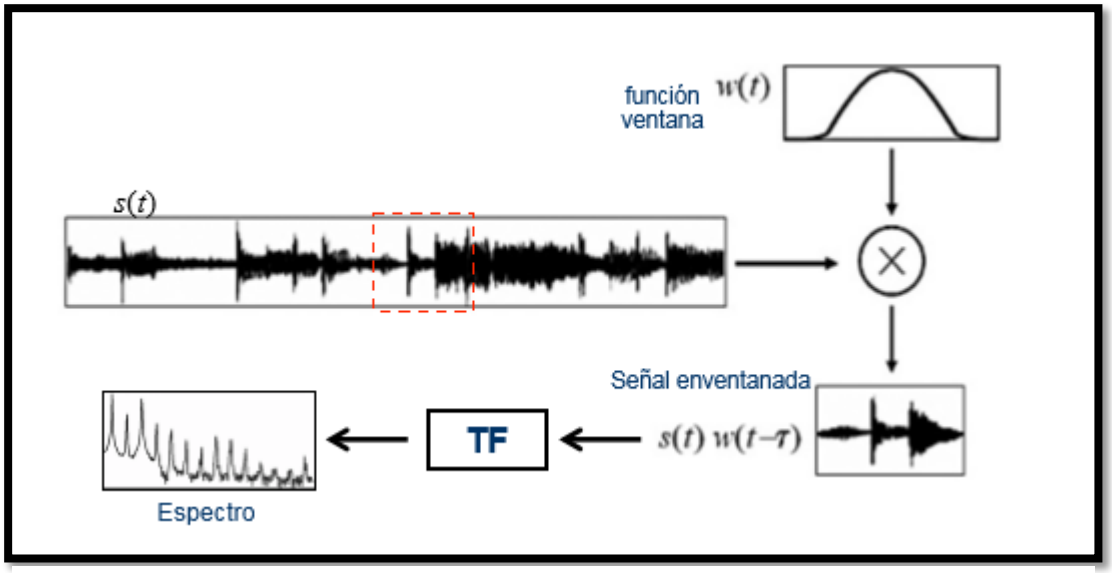


Figura 35 - Calculo del espectro con la función ventana. TF respresenta la Transformada de Fourier, que en el caso de la aplicación es la FFT.

Fuente: Diapositivas de la asignatura Sonido y Música por Computador

Como hemos dicho, la frecuencia de muestro del sistema es de 44.100 Hz, lo que significa que cada segundo se recogen 44.100 muestras. Por otra parte, las ventanas a las que se les aplica la FFT son de 1024 muestras, por lo que cada ventana representa 0.023 segundos. Debemos tener esto en cuenta a la hora de lanzar las alertas, ya que si en una ventana encontramos una frecuencia que coincide con un sonido del diccionario no debería ser suficiente para ser considerado un peligro, ya que su duración es muy corta y los peligros no presentan estas características en la realidad. Por lo tanto, el sistema se guardará el número de ventanas con posibles alertas y sólo las lanzará cuando se encuentren 4 o más con un mismo tipo de alerta en la duración de 8 ventanas. Se considera que una alerta no tiene por qué aparecer en ventanas consecutivas ya que hay factores externos que pueden influir en su frecuencia o mismamente la señal sonora del peligro puede cambiar.

Por ejemplo, en el caso de la figura 36, se lanzaría una alerta, ya que hay más de 4 positivos en un tiempo de 8 ventanas.

Figura 36 - Vector de alertas. T: true, representa un positivo para lanzar la alerta. F: false, representa ausencia de peligros.

Fuente: Creación propia

En conclusión, este tipo de filtrado se basa en obtener el espectrograma de la señal que recibe el micrófono del dispositivo y comparar sus frecuencias más representativas con las que se han analizado previamente y están en el diccionario de sonidos. Si esta comparación resulta positiva se guardará y cuando se encuentren 4 positivos en un tiempo de 8 ventanas se lanzará una alerta con el tipo de sonido encontrado.

A la hora de hacer la comparación de frecuencias debemos tener en cuenta la resolución frecuencial. Ésta representa cuantas frecuencias se encuentran agrupadas dentro de una posición del array. Es decir, si la respuesta de la FFT representa las frecuencias desde 0 hasta *fs* pero nuestro array es de N posiciones, cada posición del array agrupará *fs*/*N* frecuencias. En el caso concreto de nuestra aplicación cada posición de array tendrá 44.100 / 1024 = 43.006 frecuencias. Como hemos explicado anteriormente la frecuencia serán múltiplos de la resolución frecuencial, por lo que sólo podremos saber la magnitud de dichas frecuencias y no la de todas las frecuencias de la señal. En definitiva, si buscamos en nuestro espectro la magnitud de 400 Hz no la podremos encontrar, ya que no es un múltiplo entero de la resolución frecuencial. Por lo tanto tendremos que analizar los valores del espectro más cercanos a este, que serían 9*43.006 = 387.59 y 10*43.006 = 430.06.

6.3.5.2.1 Librerías para el filtrado

Para poder realizar el análisis de la señal del sonido que nos llega desde el exterior al micrófono del móvil se han utilizado librerías de terceros.

Primeramente se comenzó a investigar cómo realizar la Transformada Rápida de Fourier (FFT) a las muestras obtenidas. La librería **JTransform** para Java nos proporcionaba dichas funcionalidades, pero finalmente se decidió realizar el análisis con la clase **FFT.java** y **Complex.java** que nos facilita la universidad de Princeton. **[20] [21]**

7 Pruebas

La validación de todas las funcionalidades que se han descrito anteriormente la realizaremos haciendo uso de la aplicación y sometiéndola a casos prácticos y reales para comprobar su comportamiento y poder valorar si las respuestas obtenidas son las adecuadas.

- **Capturar el sonido**

  Como se puede observar en las siguientes figuras, el sistema es capaz de capturar el sonido que llega a su micrófono en un formato PCM con 16 bits de profundidad.

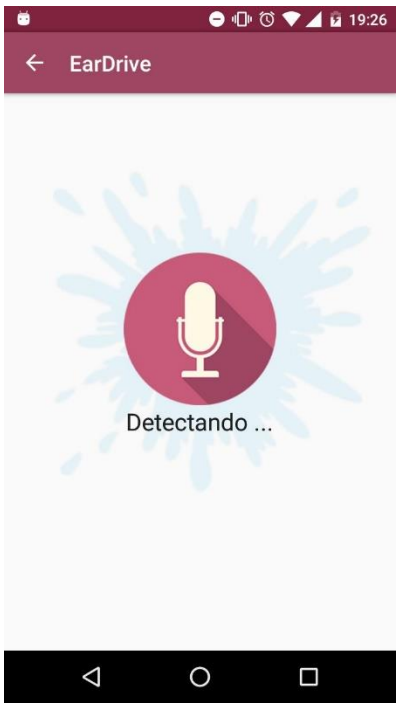


Figura 38 - Captura de pantalla del móvil cuando está capturando el sonido.

Fuente: Creación Propia

```
06-22 16:55:34.316 7201-7201/? I/SONIDO: Muestra 62 : 137
06-22 16:55:34.316 7201-7201/? I/SONIDO: Muestra 63 : 106
06-22 16:55:34.316 7201-7201/? I/SONIDO: Muestra 64 : 43
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 65 : 53
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 66 : 55
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 67 : 21
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 68 : 1
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 69 : -70
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 70 : -52
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 71 : 2
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 72 : -22
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 73 : -75
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 74 : -75
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 75 : -61
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 76 : -87
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 77 : -90
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 78 : -91
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 79 : -112
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 80 : -152
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 81 : -166
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 82 : -151
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 83 : -136
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 84 : -155
06-22 16:55:34.317 7201-7201/? I/SONIDO: Muestra 85 : -196
```

Figura 37 – Valores de las muestras obtenidas de la señal analógica que llega al teléfono.

Fuente: Creación propia

- **Obtener volumen del sonido ambiente**

  Como se ha explicado en la parte de desarrollo, el sistema capturará el volumen del sonido ambiente del interior de vehículo para

posteriormente pode realizar el filtrado por volumen. Como se observa en las figuras siguientes, la aplicación calcula los DBFs correctamente.

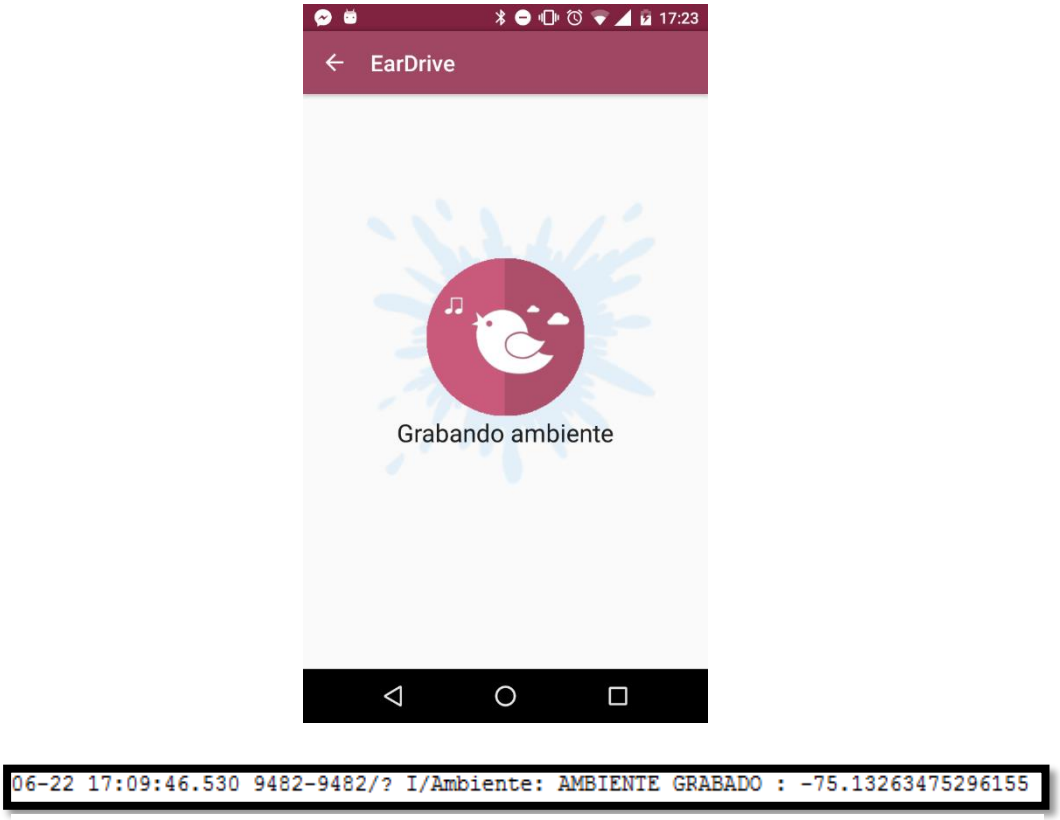


Figura 39 - Captura de pantalla del teléfono móvil grabando el sonido ambiente y salida por consola con el valor en DBFs de este.

Fuente: Creación propia

- **Alerta por volumen**

  El sistema será capaz de detectar aquellos sonidos que sobrepasen el umbral máximo definido por el usuario y lanzará una alerta visual. Esta alerta también será enviada al reloj inteligente si se encuentra emparejado con el teléfono móvil.

  En las siguientes figuras se comprueba las capturas de pantalla tanto del teléfono móvil como del reloj inteligente cuando se detecta una alerta por volumen. Además debajo se observa la salida por consola de los valores en DBFs del sonido ambiente, de la alerta y de la posición del umbral.

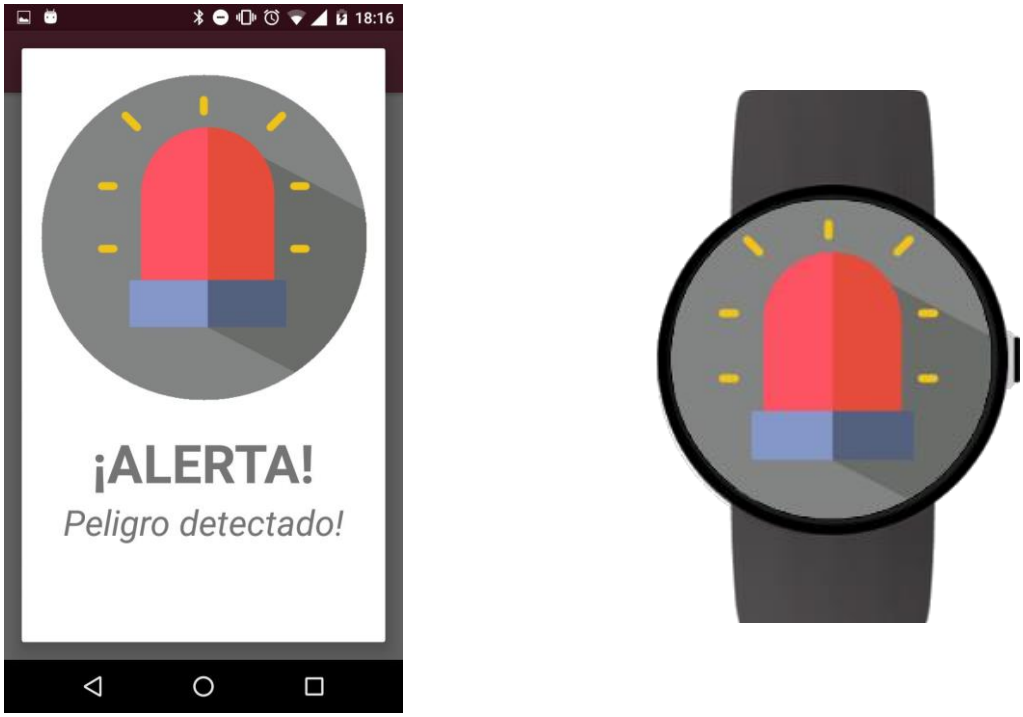


Figura 40 - Capturas de pantalla del móvil y del reloj inteligente con una alerta por volumen. Debajo se encuentra la salida por consola de la aplicación

Fuente: Creación propia

- **Alerta por claxon:**

  El sistema será capaz de detectar si suena un claxon en sus proximidades y lanzará una alerta visual en caso positivo. Al igual que en el caso de alerta por volumen, si existe un reloj inteligente emparejado al móvil también se enviará a este dispositivo.

  En las imágenes se puede observar las capturas de pantalla de los dos dispositivos así como la salida por consola de las frecuencias más altas detectadas por el sistema. Como se ha explicado en el apartado Diccionario de sonidos, la frecuencia con más energía de la señal del

claxon ronda los 1200 Hz, por lo que las salidas se corresponden con este valor.

```
06-22 18:39:20.977 30657-30657/? I/FRECUENCIA: Frecuencia : 1032.0
06-22 18:39:21.042 30657-30657/? I/FRECUENCIA: Frecuencia : 1032.0
06-22 18:39:21.092 30657-30657/? I/FRECUENCIA: Frecuencia : 989.0
06-22 18:39:21.142 30657-30657/? I/FRECUENCIA: Frecuencia : 1032.0
06-22 18:39:21.198 30657-30657/? I/FRECUENCIA: Frecuencia : 1247.0
06-22 18:39:21.266 30657-30657/? I/FRECUENCIA: Frecuencia : 1247.0
06-22 18:39:21.331 30657-30657/? I/FRECUENCIA: Frecuencia : 1247.0
06-22 18:39:21.393 30657-30657/? I/FRECUENCIA: Frecuencia : 1247.0
06-22 18:39:21.447 30657-30657/? I/FRECUENCIA: Frecuencia : 989.0
06-22 18:39:21.511 30657-30657/? I/FRECUENCIA: Frecuencia : 1032.0
06-22 18:39:21.579 30657-30657/? I/FRECUENCIA: Frecuencia : 1247.0
06-22 18:39:21.644 30657-30657/? I/FRECUENCIA: Frecuencia : 1247.0
```

Figura 41 - Salida por consola de las frecuencias con más energía detectadas.

Fuente: Creación propia.

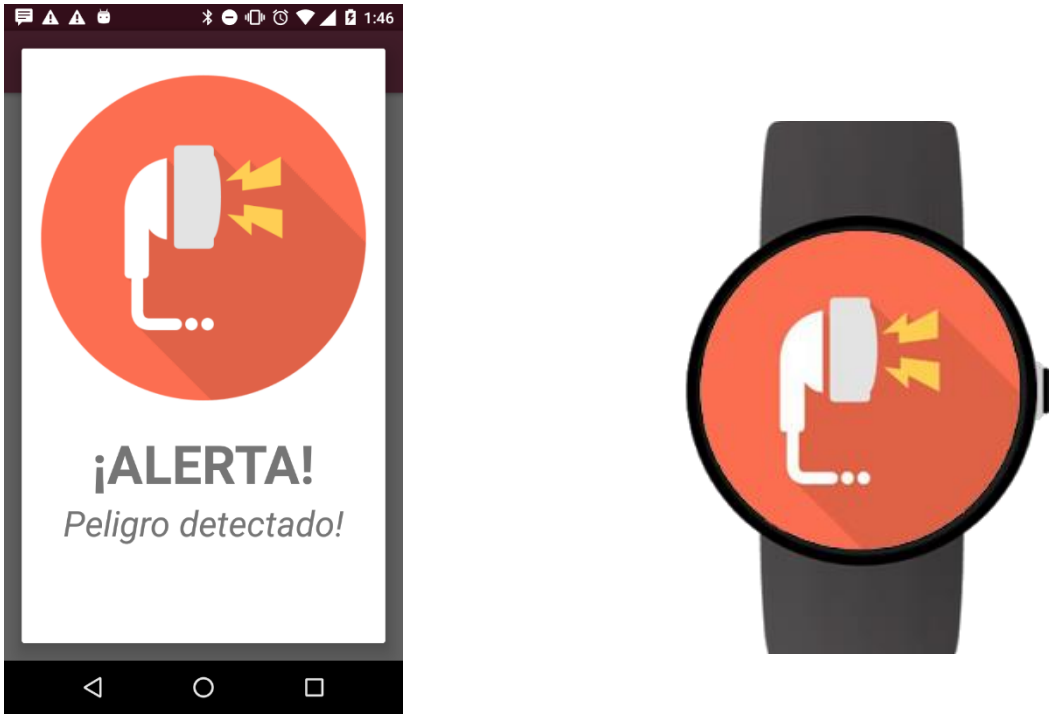


Figura 42 - Captura de pantalla del móvil y del reloj con la alerta por claxon

Fuente: Creación propia

- **Detección de la actividad del usuario**

  Cuando el usuario active esta opción el sistema debe ser capaz de detectar la actividad que se está realizando: caminar, ir en bicicleta, en un vehículo, quieto. Si la acción detectada es 'Quieto' y tiene una fiabilidad de más del 80%, el sistema parará automáticamente de grabar.

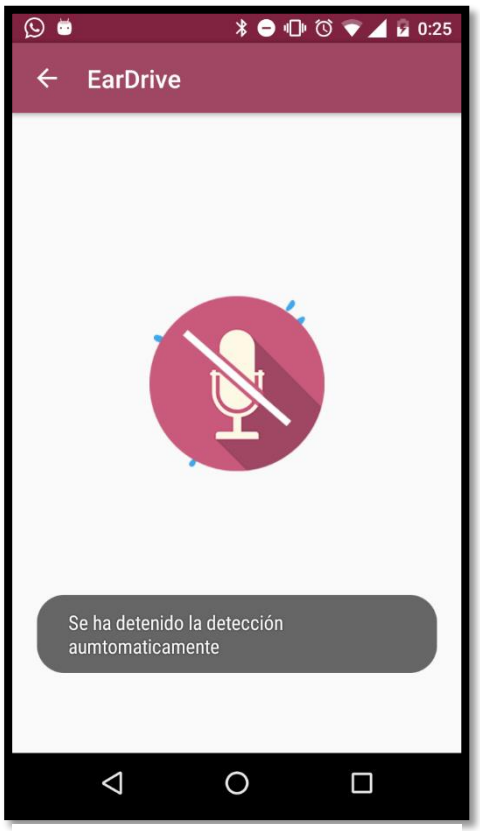


Figura 43 - Captura de pantalla de la aplicación cuando el sistema ha detectado que el usuario está quieto y deja de detectar sonidos.

Fuente: Creación propia

```
06-23 00:25:09.859 12910-12910/? I/MainActivity: Activity: Still, Confidence: 85%
                                                 Activity: In a vehicle, Confidence: 15%
```

Figura 44 - Salida por consola de las actividades detectadas por la aplicación

Fuente: Creación propia

- **Preferencias de usuario**

  El sistema deberá guardar las preferencias que el usuario modifique a su gusto en la pantalla de Configuración. Si se cierra la aplicación o el móvil es reiniciado estas deben permanecer como se configuraron por última vez.

  Esta funcionalidad cumple los objetivos marcados correctamente, su funcionamiento es el deseado.

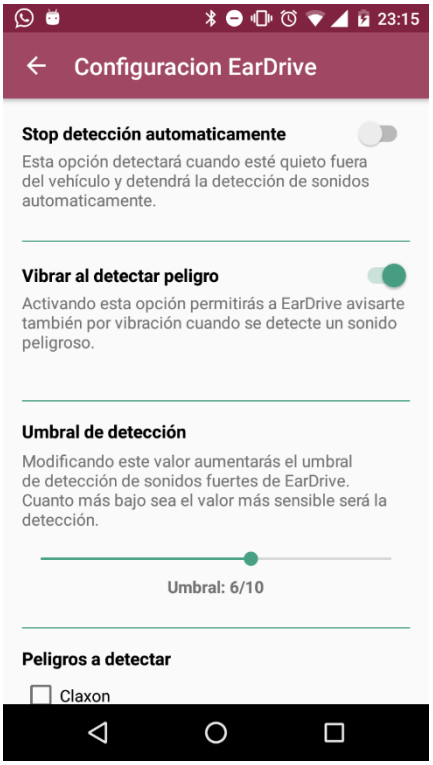


Figura 45 - Captura de la pantalla Configuración.

Fuente: Creación propia

- **Conexión con el SmartWatch**

  Ya se ha demostrado que la conexión con el reloj inteligente y que la comunicación entre ambos dispositivos funciona correctamente, pero en la aplicación se ha añadido otra funcionalidad con la cual el usuario podrá comprobar si su teléfono móvil está emparejado con algún reloj sin la necesidad de enviarle ningún mensaje de alerta.

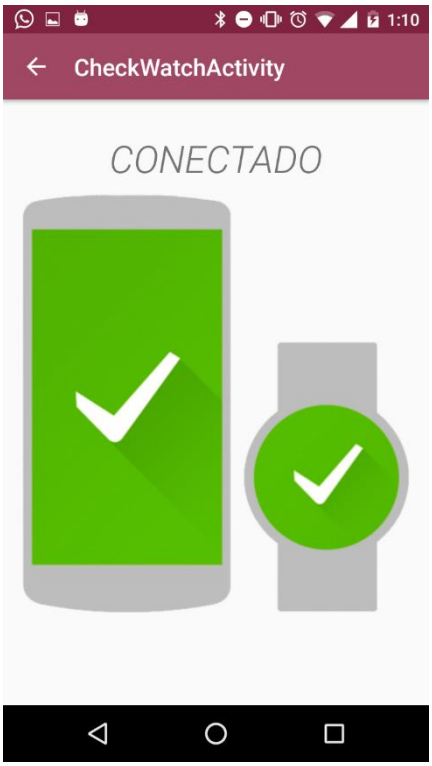


Figura 46 - Captura de la pantalla de comprobación wearable emparejado.

Fuente: Creación propia

En la sección de Anexos se dan más detalles de la implementación, mostrando tanto el resultado final del diseño de la aplicación como su estructura

8 Trabajo futuro

El sistema desarrollado para este Trabajo de Final de Grado ha conseguido los objetivos marcados en sus inicios, pero un proyecto de estas características debe estar en continua expansión y actualización, ya que a medida que avanzan las tecnologías aparecen nuevas herramientas que podrían ser de gran ayuda para el filtrado de las señales como en la comunicación con el usuario. Algunos de los aspectos que se podrían desarrollar en un futuro para incrementar el valor de la aplicación y sus funcionalidades serían:

- **Limpieza de la señal sonora recibida:** Actualmente el sistema está trabajando con la señal que le llega al micrófono en 'bruto', es decir, no se pasa ningún filtrado para mejorar su calidad y que su interpretación por los algoritmos sea más precisa. Debemos ser conscientes de que en un ambiente como el de la conducción, el ruido ambiente es muy fuerte, y por lo tanto puede tener una gran influencia en la señal que recibimos en el micrófono del teléfono móvil. De esta manera, un gran avance en el análisis de la señal sería el eliminar todo este ruido ambiente que se encuentra en nuestra nuestras muestras.

  Para poder conseguir esta 'limpieza' de la señal se podría implementar un filtro de Wiener, el cual se encargaría de reducir este ruido y por lo tanto la señal a filtrar se encontraría más cerca a la deseada.

- **Ampliar el diccionario de sonidos:** Actualmente el sistema reconoce el sonido de un claxon y aquellos sonidos que tengan un volumen alto. Una importante mejora y ampliación del sistema sería el extender y completar este diccionario de sonidos con todos aquellos que se considere que pueden representar un peligro en la conducción.

- **Compatibilidad con otros sistemas operativos:** Uno de los objetivos de la aplicación es el de generalizar el uso del sistema. Actualmente el sistema es compatible con teléfono móviles Android y relojes inteligente

Android Wear, por lo que una buena mejora para extender el uso de la aplicación sería desarrollarla también para iOS.

- **Uso del NDK de Android:** Si los algoritmos de filtrado del sistema se amplían y mejoran estos trabajarán y calcularán una gran cantidad de datos, por lo tanto el análisis en tiempo real de la señal podría verse perjudicado. Una buena idea para mejorar los tiempos de respuesta de los algoritmos sería implementarlos en C o C++ y utilizar el NDK de Android para ejecutarlos como librerías.

- **Uso de más de un micrófono:** Actualmente el sistema sólo recibe señales sonoras por una entrada, el micrófono del dispositivo móvil. Una buena ampliación del sistema sería el utilizar más de un micrófono para mejorar la calidad del sonido que analizamos. Por otra parte, se podría mejorar la información que se le da al usuario ya que el sistema podría detectar dónde está el peligro y avisar de la posición (izquierda, derecha, delante o detrás) de la alerta auditiva.

9 Conclusiones y valoración personal

Tras finalizar este Trabajo de Fin de Grado cabe destacar que el resultado es del todo satisfactorio para su autor. Los objetivos marcados en un inicio fueron completados prácticamente en su totalidad, y el resultado de tanto trabajo y esfuerzo es un prototipo funcional que sienta las bases de una aplicación que puede ser útil a un sector desfavorecido de la sociedad.

Se ha realizado una aplicación efectiva, con un tiempo de respuesta y de ejecución rápido y óptimo. La comunicación con el reloj inteligente es fluida y prácticamente instantánea, la demora en la entrega del mensaje es imperceptible para el usuario. Además, se ha minimizado el tamaño de la aplicación, consiguiendo reducir su peso a 4 MB aproximadamente.

Asimismo, se ha conseguido la compatibilidad con la mayoría de las distribuciones Android, ya que la mínima soportada por el sistema es Android 2.3 o Android 4.3 en el caso de que se desee incluir el reloj inteligente en el funcionamiento de la app. Se ha implementado también una aplicación para "wearables" totalmente funcional y compatible con el SO Android Wear.

Al mismo tiempo, los resultados obtenidos en el diseño de la aplicación son los esperados. Se ha conseguido un diseño usable, adaptativo y que con sus características permita a la aplicación no interferir en la conducción del vehículo.

Por otra parte se debe mencionar que la planificación temporal establecida en un inicio no fue cumplida. Esto es debido a que etapas como las del diseño de la aplicación y sobre todo el análisis de la señal llevaron más tiempo del previsto inicialmente.

Para el desarrollo de este proyecto se han aplicado todos los conocimientos adquiridos durante los cuatro años de carrera en Ingeniería Multimedia, así como también se ha recurrido a la investigación y al aprendizaje por parte propia en aquellas materias en las que no se poseía de los conocimientos necesarios. Una de las áreas en las que más tiempo se tuvo que invertir en aprendizaje e investigación ha sido en la del análisis y procesamiento de las señales sonoras. Cierto es que durante el Grado existen

asignaturas que me proporcionaron los conocimientos fundamentales sobre dicha materia, pero este tipo de algoritmos tan específicos y complejos requerían de una formación más profunda y completa.

Cabe destacar que uno de los aspectos más importantes y que más me complacen al terminar este TFG es el haber ampliado mi formación y adquirido nuevos conocimientos en materia de diversidad funcional y accesibilidad. Considero que en un mundo donde las tecnologías comienzan a generalizar su uso, se debe fomentar y formar la investigación es este tipo de áreas, ya que simples herramientas como esta aplicación podrán facilitar el día a día de estas personas y brindarles nuevas oportunidades.

10 Referencias

[1] «**La Sociedad en Red**» http://www.abc.es/gestordocumental/uploads/internacional/NP%20RED.ES_%20Informe%20ONTSI%2024%2007%2014.pdf

Ultimo acceso Junio del 2016

[2] **Conducción con discapacidad auditiva -** http://www.centrouribarri.es/conduccion-con-discapacidad-auditiva/

[3] **Audio Aware -** https://www.tecnocarreteras.es/2014/03/14/una-aplicacion-que-reconoce-aquellos-sonidos-que-presagian-peligro-para-alertar-al-usuario-despistado/

Ultimo acceso Abril del 2016.

[4] **My Ear Droid -** http://www.tecnalia.com/es/myeardroid/que-es-myeardroid.htm

Ultimo acceso Abril del 2016.

[5] **BioAid -** http://www.gizmag.com/bioaid-iphone-hearing-aid-app/26857/

Ultimo acceso Abril de 2016

[6] **¿Qué es iOS?** - http://conceptodefinicion.de/ios/

Ultimo acceso 28 de Junio del 2016.

[7] **iOS** - http://www.actualidadiphone.com/category/ios/

Ultimo acceso Junio de 2016.

[8] **Que es Windows Phone** - http://conceptodefinicion.de/windows-phone/

Ultimo acceso Junio de 2016.

[9] **¿Qué es Android? -** http://www.xatakandroid.com/sistema-operativo/que-es-android

Ultimo acceso Abril de 2016.

[10] **Get Start, Android -** https://developer.android.com/about/start.html

Ultimo acceso Abril de 2016.

[11] **MarshMallow -** http://www.elandroidelibre.com/2015/08/si-lollipop-fue-diseno-marshmallow-es-rendimiento.html

Ultimo acceso Abril de 2016.

[12] **Android Wear -** https://es.wikipedia.org/wiki/Android_Wear

Ultimo acceso Abril de 2016.

[13] **Java (lenguaje de programación)** - https://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29

Ultimo acceso Mayo de 2016

[14] **XML -** https://www.ecosia.org/search?q=xml&region=&lang=&f=false

Ultimo acceso Mayo de 2016

[15] **Material Design -** http://www.elandroidelibre.com/2014/11/que-es-material-design.html

Ultimo acceso Junio de 2016.

[16] **Frecuencia de muestreo -** https://es.wikipedia.org/wiki/Frecuencia_de_muestreo

Ultimo acceso Junio de 2016

[17] **AudioRecord -** https://developer.android.com/reference/android/media/AudioRecord.html

Ultimo acceso Junio de 2016

[18] **Psicoacústica y Sonoridad** – Apuntes de la asignatura de Sonido y Música por Computador de Ingeniería Multimedia.

[19] **Bases físicas y matemáticas del sonido –** Apuntes de la asignatura de Sonido y Música por Computador de Ingeniería Multimedia. Pag.20.

[20] **Clase FFT.java** - http://introcs.cs.princeton.edu/java/97data/FFT.java.html

Ultimo acceso Junio de 2016.

[21] **Clase Complex.java** - http://introcs.cs.princeton.edu/java/97data/Complex.java.html

Ultimo acceso Junio de 2016.

11 Anexos

En este apartado se detallarán el resultado final del diseño de la aplicación y también se explicará la estructura del proyecto para posibles ampliaciones futuras.

11.1 Resultado final del diseño de la aplicación

Se reflejará cual ha sido el resultado final del diseño de la aplicación a través de capturas de pantalla de todas las secciones de la app.

- **Pantalla de inicio**

Figura 47 - Captura de la pantalla de inicio de la aplicación.

Fuente: Creación propia

- **Pantalla de opciones de usuario**

  Esta sección estará superpuesta a la pantalla de inicio. Se puede acceder a ella presionando sobre el botón flotante con el símbolo más (+) que aparece en la parte inferior derecha de la pantalla de inicio.

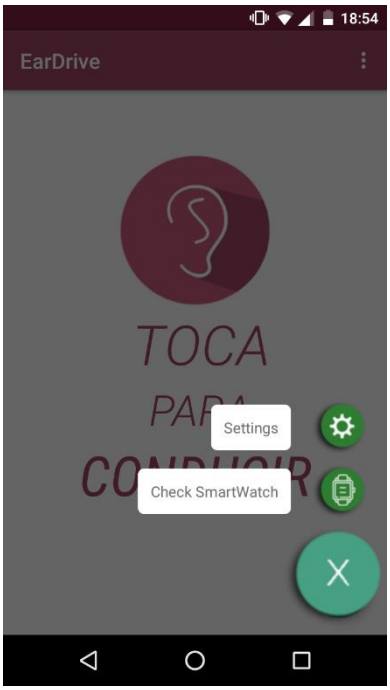


Figura 48 - Captura de la sección de opciones de usuario.

Fuente: Creación propia

- **Pantalla de configuración.**

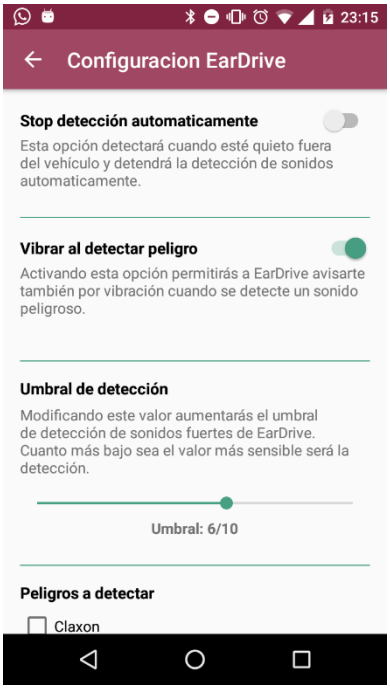


Figura 49 - Captura de la pantalla configuración

Fuente: Creación propia

- **Pantalla para comprobar si tenemos un Smartwatch emparejado**

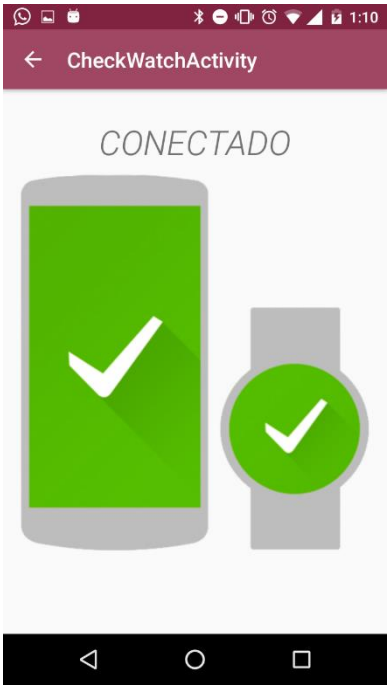


Figura 51 - Captura de pantalla de comprobación de Smartwatch

Fuente: Creación propia

Figura 50 - Captura de pantalla con Smartwatch no conectado

Fuente: Creación propia

- **Pantalla Detectando**

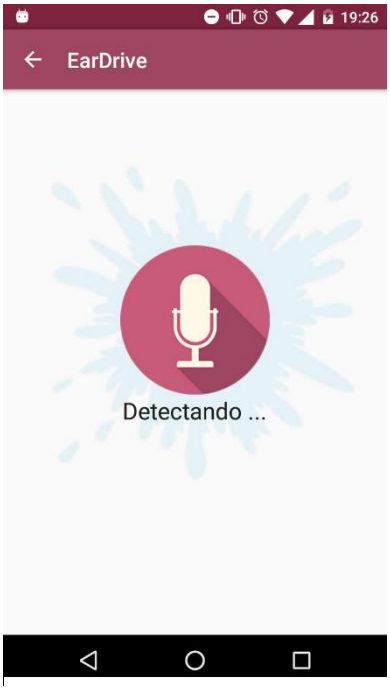


Figura 52 - Captura de la pantalla Detectando

Fuente: Creación propia

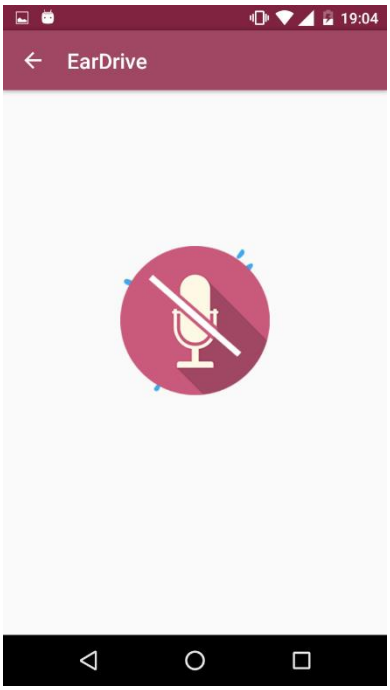


Figura 53 - Captura de pantalla Detectando cuando se ha parado la detección.

Fuente: Creación propia

- **Alertas**

Figura 55 - Captura de Alerta por sonido fuerte.

Fuente: Creación propia

Figura 54 - Captura de Alerta por claxon.

Fuente: Creación propia

- **SmartWatch**

Figura 57 - Pantalla inicio en el reloj inteligente

Fuente: Creación propia

Figura 58 - Pantalla alerta claxon en el reloj inteligente.

Fuente: Creación propia

Figura 56 - Pantalla de alerta por sonido fuerte en reloj inteligente

Fuente: Creación propia.

11.2 Guía de desarrollo

En esta sección del documento se explicará más a fondo cual es la estructura del proyecto, donde se implementan cada una de las funcionalidades y los métodos que se han utilizado para su desarrollo.

El objetivo de esta sección es facilitar el aprendizaje y la toma de contacto con el sistema para aquellas personas que deseen continuar trabajando y ampliando *EarDrive*.

Se irán explicando las Actividades de la aplicación ya que en ellas es donde reside todo el comportamiento del sistema.

11.2.1 Inicio Activity

Esta Activity es la primera que lanza el sistema cuando comienza su ejecución. La interfaz (layout) asociada a dicha actividad es Activity_inicio.xml y content_inicio.xml.

En esta actividad únicamente se implementa un controlador de eventos sobre la pantalla, *onTouchEvent(MotionEvent event)*, el cual se encargará de detectar cuando el usuario toca la pantalla y lanzará la siguiente Actividad, que en este caso sería DetectandoActivity.

Además, en esta actividad se controlará cuando el usuario presione sobre el botón para abrir las opciones de usuario, *FloatingActionButton*, y se abrirá el menú.

11.2.2 Configuracion Activity

Esta Activity se lanzará desde la pantalla de inicio cuando el usuario acceda a las opciones y presione sobre 'Settings'. El layout asociado a dicha Activity es activity_configuracion.xml y content_configuracion.xml.

Esta Actividad es la encargada de mostrar todas las opciones que el usuario puede modificar y ajustar a su gusto. Se implementan por lo tanto, un slider con el elemento SeekBar de Android para controlar el umbral de detección, elementos Switch para activar o desactivar opciones como la vibración y la detección de actividad y por último, se hace uso de CheckBoxes para seleccionar los tipos de sonidos que el usuario quiere filtrar.

Cada uno de estos elementos tendrá un controlador que manejará sus cambios de valores. En estos controladores se accederá a las preferencias del usuario a través del objeto *SharedPreferences* y se guardará el nuevo valor seleccionado.

11.2.3 CheckWatch Activity

Esta Actividad será también lanzada desde la pantalla de inicio a través del menú de opciones de usuario. El layout asociado a dicha Actividad es activity_check_watch.xml y contet_check_watch.xml

Aquí el sistema comprobará si existe algún reloj emparejado con el teléfono móvil. Para ello utilizaremos los Servicios de Google Play y su Api para wearables. Por lo tanto iniciaremos un cliente de GoogleApiClient() con la Api WearableAPI.

Una vez nuestro cliente se encuentre conectado procederemos a hacer uso de NodeAPI para listar los dispositivos emparejados con el teléfono móvil. Si esta lista no se encuentra vacía ya sabremos que el dispositivo móvil tiene un wearble asociado y se lo podremos comunicar al usuario a través de la interfaz gráfica.

11.2.4 Detectando Activity

Esta Activity se lanzará cuando el usuario toque sobre la pantalla de inicio. Su layout asociado es activity_detectando.xml y content_detectando.xml.

Es la Actividad principal del sistema. Con ella se controlará el filtrado de la señal sonora, se detectarán las actividades que está realizando el usuario y se lanzarán las alertas.

Para capturar la señal y realizar su filtrado se hará uso de la clase SoundMeter.java. Se creará un objeto de dicha clase y se accederá a sus métodos. SoundMeter es la encargada de acceder al micrófono del dispositivo y capturar el audio que éste graba. Esto se realizará con la clase AudioRecord que Android nos proporciona. Será en esta clase también donde se realicen los dos tipos de filtrado que se han explicado en el apartado de Desarrollo. Para el filtrado por volumen se utilizará al método *getAmplitud*(), mientras que para el filtrado por frecuencia se utilizarán las clases FFT y Complex para poder operar con la transformada de Fourier y obtener el espectrograma. Todo esto se realiza en los métodos *calculateFFT()* y *CompararFrecuencias()*.

Cabe destacar que desde Detectando Activity se crea un nuevo hilo (mPollTask Runnable) para hacer las llamadas a SoundMeter en segundo plano. Estas llamadas se realizan cada 23 milisegundos dado que es el tiempo que tarda la ventana de 1024 muestras en completarse. Desde este hilo se llamará a los métodos *getAmplitud()* y *comprobarAlerta()* y se verificará si se debe lanzar una alerta. En caso positivo se llamará al método *darAlerta(tipo)*.

Esta Actividad es por lo tanto, la encargada de lanzar el alertDialog de las alertas. Este dialogo aparecerá implementado en la Activity como una clase privada de la cual se creará un objeto y se destruirá en la medida que el sistema lo requiera. Los métodos

utilizados para esto serán: *showDialog(tipoAlerta)* y *dismisDialog().* Además, cuando se lanza una alerta se debe enviar también a todos los dispositivos wearables que se encuentren conectados al teléfono. Para esto se hace uso de la WearableAPI de Android y su MessageAPI. En el método *sendMessage(tipoAlerta)* se creará otro hilo para que el proceso del envío del mensaje se realice también en segundo plano.

Por otra parte, si el usuario ha configurado que desea que el sistema detecte sus Actividades, se debe hacer uso de los Servicios de Google Play y de su API ActivityRecognition. Para esto se hará que la clase implemente las interfaces *ConnectionCallbacks y OnConnectionFailedListener*. Se creará también un cliente de GoogleAPI y se añadirá ActivityRecognitionAPI. Además para poder reconocer la Actividad que está realizando el usuario se utilizarán la clase *Constants.java*, donde se almacenarán datos que la API utilizará y el servicio al que llamará la API una vez sea reconocida la Actividad del usuario, *ActivityRecognizedService.java.* Se deben implementar lo métodos *onConnected*(), que será al que se llame cuando se realice la conexión con los Servicios de Google Play, *onConnectionSuspended(),* que se llamará cuando se suspenda la conexión y *onConnectionFailed()* que se llamará cuando falle la conexión. Será en el método onConnected donde se realicen las peticiones de actualización de las actividades del usuario.

Se manejará en esta clase también la petición de permisos en tiempo de ejecución incluida con Android 6.0. Esto se realizará en el método *onResume()* de la aplicación y haciendo uso de *requestPermision()*.

11.2.5 Comportamiento del Smartwatch

En el reloj inteligente solamente habrá una actividad y un ListenerService que escuchará los mensajes que se le envíen desde el teléfono móvil.

- **WearMessageListenerService**

  Será un servicio que hereda de la clase *WearableListenerService.* Sobrescribiremos el método *onMessageRecived(),* en el cual se manejará el mensaje que se recibe a través de *MessageAPI,* y si es el mensaje correcto se lanzará la Actividad principal de la aplicación wearable.

- **InicioActivity**

  Es la actividad principal de la aplicación wearable. El layout asociado a dicha actividad es activity_inicio.xml y rect o round _activity_inicio.xml.

  En ella únicamente recibiremos el tipo de alerta que nos ha enviado el ListenerService y en consecuencia a dicho tipo mostraremos por pantalla un icono u otro. Además se lanzará la vibración del dispositivo.