UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

UNIVERSIDAD
POLITECNICA
DE VALENCIA

ESCUELA POLITECNICA
SUPERIOR DE GANDIA

CREACION DE UNA APP MOVIL PARA ANDROID CAPAZ DE DETECTAR ULTRASONIDOS.

TRABAJO FINAL DE GRADO

Autor:
Ángel Pozo Puchalt

Tutor:
Paco Camarena

Cotutor:
Gabriel Moreno Ibarra

GANDIA, 2017

Resumen

En esta memoria se expone y documenta la creación de una APP para Android, la cual detecta ondas de ultrasonido e indica proximidad a la fuente emisora, además de un producto o marca asociados a dicha señal acústica.
Para llevar a cabo este proyecto se utilizan diferentes librerías externas, diferente software para las diversas tareas, se realiza un estudio del consumo de la batería a causa de diferentes modos de la aplicación y se realiza un estudio acústico para comprender que rango de frecuencias es capaz de emitir un altavoz convencional, así como que micrófonos pueden recibir dichas frecuencias mencionadas.
Es necesario el conocimiento de conceptos adquiridos en la ingeniería de sistemas de telecomunicaciones, imagen y sonido para la comprensión y seguimiento de este documento.
Respecto a los resultados, se comprueba que la detección es satisfactoria a diversas distancias, y con diferentes equipos. En el apartado correspondiente se detallan las pruebas realizadas, junto a los equipos utilizados y los resultados finales. En general, el resultado es positivo y se encuentra en la aplicación una manera práctica y novedosa de realizar compras o de facilitar al usuario información concreta sin necesidad de realizar búsquedas en los buscadores web.

Abstract

This document describes and explains the creation of an APP for Android, which detects ultrasound waves and indicates proximity to the source, as well as a product or brand associated to the acoustic signal.
In order to carry out this project, different external libraries are used. Different software for different tasks. A study of the consumption of the battery is made because of different modes of the application. An acoustic study is carried out to understand which frequency range is capable of emitting a conventional loudspeaker, as well as which microphones can receive said mentioned frequencies.
It is necessary the knowledge of concepts acquired in the engineering of telecommunications systems, image and sound for the understanding and monitoring of this document.
Regarding the results, it is verified that the detection is satisfactory at different distances, and with different equipment. The corresponding section details the tests performed, together with the equipment used and the final results. In general, the result is positive and is in the application a practical and novel way to make purchases or provide the user with specific information without having to search the web search engines.

Palabras clave

Ultrasonidos, aplicación, smartphone, Android, IOS, batería, dispositivo, detección, marketing, ventas, XTAudioBeacion, producto, servicio, escucha, actividad, funcionalidad, usuario.

Keywords

Ultrasound, application, smartphone, Android, IOS, battery, device, detection, marketing, sales, XTAudioBeacon, Product, service, listening, activity, functionality, user.

A los que creyeron, a los que estuvieron ...

"Pensar sin aprender es esfuerzo perdido; aprender sin pensar, peligroso".

Confucio.

Agradecimientos

A GitHub y Google, fuente de toda sabiduría.

A mi familia, por hacer esto posible.

A mis amigas y amigos, por creer en mí, más de lo que yo creí.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Motivación

La creación de una app para Android no es ninguna novedad y cada vez resulta más complicado ser innovador dentro de este ámbito.

Por otro lado, hoy en día, todas las empresas necesitan tener su propia aplicación móvil para la venta o gestión de sus productos y servicios. Parece que si no tienes tu app no existes en el mercado.

Una de las intenciones del proyecto es, intentar unificar compras de varias empresas. Es decir, liberar al usuario de tener que gestionar las compras de productos de cada marca con su respectiva aplicación móvil, consiguiendo así una mayor comodidad para realizar compras o acceder a información muy concreta.

La innovación viene cuando se utiliza el rango de frecuencias conocido como ultrasonido para despertar una posible compra online en los Smartphone Android.

Para hacer esto posible, el Smartphone del usuario debe encontrarse cerca de un emisor, en este caso cualquier altavoz convencional servirá. Cuando el usuario escuche o visualice información sobre un producto o servicio, podrá obtener más información en su móvil simplemente abriendo la aplicación y detectando la huella digital insertada previamente en el anuncio.

Si te encuentras en cualquier tienda física, o cerca de ella, desde grandes supermercados, centros comerciales o pequeñas empresas, puedes detectar las ofertas de ese momento, día o temporada, sin tener que estar informado previamente ni tener que estar pendiente de carteles publicitarios, etc.

Esto se pretende conseguir únicamente con un altavoz convencional que realice funciones de emisor y un Smartphone que será el receptor.

Para conseguirlo se emite un audio que actuará como huella digital, a unas frecuencias comprendidas entre 16 kHz y 22 kHz.

1.2 Objetivos

El principal objetivo de este Trabajo de Fin de Grado es el de elaborar de principio a fin una aplicación para dispositivo móvil en Android, desde la planificación y diseño, hasta el desarrollo y lanzamiento, que permita conseguir un aprendizaje de los procesos que deben cumplirse desde que se tiene una idea general de lo que se quiere crear hasta que esa idea coge forma y se convierte al final de todas las etapas en una aplicación disponible en Play Store al alcance de cualquier usuario y perfectamente funcional.

Por ello, se pretende crear una aplicación para dispositivos Android capaz de ofrecerte productos dependiendo de lo que el usuario escucha o en el lugar donde se encuentre.

Cuando el usuario active dicha aplicación, esta deberá detectar un XTAudio, el cual estará asociado a una única oferta/producto. El usuario visualizará la oferta, pudiendo omitirla o comprarla. Además, tendrá la opción de añadirla a favoritos en el caso de que quiera decidirse más tarde. Todo esto se debe conseguir sin la conexión a internet con excepción de realizar la compra, ya que se redirige a la web oficial de la marca.

Otro de los principales objetivos de un producto de estas características es la forma de llegar al usuario, implantando una interfaz amigable e intuitiva que permita la facilidad de uso.
Por otra parte, se realiza un estudio de fiabilidad respecto al consumo de la batería estando la aplicación en constante escucha.

Para ello, es necesario conseguir realizar la detección de esos audios y contar con una base de datos donde poder consultar cada huella digital.

En el modo automático, es posible que existe el hándicap, de no tener un correcto funcionamiento de la aplicación por llevar el Smartphone en los bolsillos de los pantalones, bolsos o metidos en algún lugar similar. Se realizará un estudio para comprobar si los XTaudio consiguen penetrar diferentes tejidos.

Capítulo 2. Estado del arte

En cuanto a la historia de la tecnología utilizada, *Google* adquiere *Android Inc.* en el año 2005. Se trataba de una pequeña compañía, recién creada, orientada a la producción de aplicaciones para dispositivos móviles. Ese mismo año, se empieza a trabajar en la creación de una máquina virtual *Java* optimizada para móviles (*Dalvik VM*).

En el año 2007 se crea el consorcio *Handset Alliance* con el objetivo de desarrollar estándares abiertos para móviles. Está formado por *Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel, etc*. Uno de los objetivos principales de esta alianza es promover el diseño y difusión de la plataforma *Android*. Sus miembros se han comprometido a publicar una parte importante de su propiedad intelectual como código abierto bajo licencia *Apache v2.0.*

En noviembre de 2007 se lanza una primera versión del *Android SDK*. Tras un año, aparece el primer dispositivo móvil con *Android* (*T-Mobile G1*). En octubre, *Google* libera el código fuente de *Android*, principalmente bajo licencia de código abierto Apache (licencia GPL v2 para el núcleo). Ese mismo mes se abre *Android Market*, para la descarga de aplicaciones. En abril de 2009, *Google* lanza la versión 1.5 del *SDK*, que incorpora nuevas características como el teclado en pantalla. A finales de 2009 se lanza la versión 2.0 y a lo largo de 2010, las versiones 2.1, 2.2 y 2.3.

Durante el año 2010, *Android* se consolida como uno de los sistemas operativos para terminales móviles más utilizados, con resultados proóximos a *iOS* e incluso superando al sistema de *Apple* en EE.UU.

En el año 2011 se lanza la versión 3.x (*Honeycomb*), específica para tablets, y la 4.0 (*Ice Cream Sandwich*), tanto para móviles como para tablets. Durante ese año, *Android* se consolida como la plataforma para smartphones más importante y alcanza una cuota de mercado superior al 50%.

En 2012, *Google* cambia su estrategia en su tienda de descargas online, reemplazando *Android Market* por *Google Play Store*, donde en un solo portal unifica tanto la descarga de aplicaciones como la de contenidos. Ese año aparecen las versiones 4.1 y 4.2 (*Jelly Bean*). *Android* mantiene su espectacular crecimiento y alcanza, a finales de año, una cuota de mercado del 70 %.

En 2013 se lanzan las versiones 4.3 y 4.4 (*KitKat*). En 2014 se lanza la versión 5.0 (*Lollipop*). A finales de ese año, la cuota de mercado de *Android* llega al 85 %.

En octubre de 2015 ha aparece la versión 6.0, con el nombre de *Marshmallow*.

En la actualidad está establecida la versión 7.0 llamada *Nougat* desde finales del 2016.

Cada versión de *Android* lleva asociado un nombre se elige en orden alfabético (en Ingles) y está asociado a un dulce que es su logotipo correspondiente.

Capítulo 3. Metodología de trabajo

En este apartado se detalla la organización seguida para el desarrollo del trabajo de final de grado así con los tiempos empleados en cada tarea y las personas involucradas para su realización.

3.1 Metodología

Para la realización del proyecto se va a seguir una metodología ágil. Se ha elegido utilizar este tipo de metodologías para poder presentar la aplicación con su funcionalidad principal bien implementada y posteriormente se añaden diferentes funcionalidades interesantes para una mejor experiencia de usuario. Se decide seguir esta metodología porque se entiende que un proyecto de este tipo siempre puede mejorarse, hacerlo más eficiente o añadir posibles usos que el usuario pueda darle.
Con esto, se consigue centrarse en el objetivo principal y no intentar abarcar muchas utilidades que quizá a la larga no tengan mucho sentido o no se le den uso.

3.2 Documentación y planificación

Para empezar, se llevó a cabo una reunión, donde se especificaba que debería hacer la aplicación y cuáles eran las funcionalidades que debía tener.

Se investigó sobre la parte acústica del proyecto, y sobre las ondas ultrasónicas que se deberían detectar.

En la parte tecnológica, se realizó un curso básico pero completo de Android facilitado por desarrolladores expertos de la empresa Everis Spain.

En cuanto a la planificación, se realizó un esquema general donde se organizaban las tareas, desde el diseño inicial hasta el lanzamiento de la app en Play Store o Markets alternativos.

Para optimizar los tiempos y ajustarse al plazo de entrega se siguen las siguientes metodologías:

- Ágil
- MVP
- MoSCoW

3.3 Ágil

Metodología de trabajo que adapta la forma de trabar a las condiciones que el proyecto requiere consiguiendo flexibilidad para poder cumplir con plazos de entrega incluso si estos se adelantan o retrasan en el tiempo por cualquier tipo de circunstancia. Además, este método, consigue reducir costes e incrementar la productividad de las empresas.

Todo esto, se traduce en una mejor satisfacción por parte del cliente, ya que no debe esperar a ver el resultado final, si no que va viendo su evolución en los plazos marcados previamente entre cliente y empresa. También tiene la ventaja de que el

cliente pueda opinar en cada plazo y poder decidir un cambio que será más sencillo de solucionar en dicho momento que cuando todo el proyecto esté terminado.

Resumiendo, es bueno para todas las partes, la empresa reduce costes e incrementa la productividad, los desarrolladores encuentran una motivación cuando consiguen terminar llegar a su objetivo en cada plazo y el cliente percibe que se está realizando el trabajo por el cual ha pagado y puede conseguir un proyecto con el que estar más contento y de mayor calidad.

3.4 MVP

MVP (Minimum Viable Product), que traducido al castellano es Producto Viable Mínimo. Es la versión de un nuevo producto que permite recolectar, con el menor esfuerzo posible, teniendo solamente aquellas funcionalidades que permiten que el producto sea lanzado y la máxima cantidad de conocimiento validado sobre sus potenciales clientes.

Existen dos tipos de MVP:

1. Validando el clásico MVP, que consiste en aproximarse a clientes potenciales e intentar venderles un producto que aún no existe, si su respuesta es positiva, se ha alcanzado el éxito y en caso contrario no significa que se haya fracasado. Un claro ejemplo es el video Digg para Dropbox de Drew Houston en marzo de 2008, generando 70K inscripciones para un producto que no había sido lanzado aún. Un fallo de esta prueba no necesariamente significaría que Dropbox no funcionaría, podría significar que el umbral de calidad era mayor de lo esperado, o que los usuarios de Digg no eran los primeros clientes ideales, o que algún otro evento de noticias había distraído a los usuarios de ver o compartir el video. Independientemente, el fracaso de su validación MVP sólo invalida una pequeña forma de adquirir clientes, con un video en Digg, no todo el modelo de negocio.

2. Invalidando el MVP conserje o Mago de Oz, al caer en que se crea un producto mejor a un coste insostenible al personalizarlo para cada cliente. Es hora de seguir adelante si no se puede lograr que la gente pague un precio realista a largo plazo por un producto mejor. Jennifer Hyman y Jennifer Fleiss comenzaron Rent the Runway con este enfoque: proporcionaron un servicio de alquiler de vestidos en persona donde los estudiantes universitarios podían probar vestidos antes de alquilarlos, una experiencia mucho mejor que el alquiler en línea. Si nadie hubiese alquilado esa noche, habrían creído que el alquiler en línea no merecería la pena. Pero el 34% (y luego el 75%) de las mujeres alquiló, por lo que pasaron a validando un MVP, donde el 5% de 1.000 mujeres de su lista de correo alquiló vestidos de un PDF por correo electrónico.

Aplicándolo a este proyecto, validando el MVP consistía en una app con la única función de detectar sonido ultrasónico, por su innovación.

Invalidando el MVP residió en planificar por funcionalidades, es decir hasta que no estuviese funcionando correctamente la detección no se continuaba a implementar la siguiente. Como se ha comentado la más importante era la detección, luego se continuo por realizar una vista de favoritos, y continuando hasta que el plazo de entrega finalizó

Se optó por esta manera, ya que esto permitía tener una estructura de un producto que funcionase constantemente, empezando por la función con mayor potencial y terminando por añadir funciones extra.

3.5 MoSCoW

El método MoSCoW es una técnica de priorización de requisitos basada en el hecho de que, aunque todos los requisitos se consideren importantes es fundamental destacar aquellos que permiten darle un mayor valor al sistema, lo que permite enfocar los trabajos de manera más eficiente.
Lo que la diferencia de otras técnicas tradicionales como por ejemplo calificar los requisitos como de prioridad alta, media o baja es que en este caso la escala utilizada tiene un significado intrínseco, de manera que el usuario responsable de asignar la prioridad conoce el efecto real que producirá su elección.

M (Must): Requisito que tiene que estar implementado en la versión final del producto para que la misma pueda ser considerada un éxito.

S (Should): Requisito de alta prioridad que en la medida de lo posible debería ser incluido en la solución final, pero que llegado el momento y si fuera necesario, podría ser prescindible si hubiera alguna causa que lo justificara.

C (Could): Requisito deseable pero no necesario, se implementaría si hubiera posibilidades presupuestarias y temporales.

W (Won't): Hace referencia a requisitos que están descartados de momento pero que en un futuro podrían ser tenidos de nuevo en cuenta y ser reclasificados en una de las categorías anteriores.

Esta clasificación puede ser modificada durante el proceso de desarrollo y definirse, en el caso de desarrollos iterativos incrementales, prioridades a nivel de iteración.
Resumiendo, junto con el cliente, se asigna una prioridad a cada requisito en cada plazo de entrega para que el proyecto cumpla las expectativas mínimas que exige el cliente y posteriormente poder implementar funcionalidades extra que le puede dar al proyecto un valor añadido.

3.6 Gestión del proyecto

El proyecto se ha gestionado por Ángel Pozo Puchalt, como estudiante de la *Universidad Politécnica de Valencia* desde las oficinas de Madrid de *Everis Spain S.L.U.*

Para llevar a cabo este trabajo, ha sido necesaria la ayuda:

- Profesores UPV:
    - José Marín Roig. Asesor.
    - Paco Camarena. Tutor del TFG.
    - Gabriel Moreno Ibarra. Cotutor del TFG.

- Compañeros Departamento *Digital Experience Everis Spain*:
    - Santiago Santamaria. Gerente.
    - Estefanía Riera. Diseñadora de apps.
    - Jose Luis Hurtado. Desarrollador Senior.

- Ricardo Martin Blanco. Becario.
- Jose Javier Montes Romero. Desarrollador junior.

3.7 Distribución de tareas

Las tareas las distribuye el estudiante, siendo aconsejado por las personas citadas anteriormente.
En términos generales y de manera inicial se marcan los siguientes tiempos:

DISEÑO FUNCIONAL Y DISEÑO TÉCNICO (será parte de la memoria del TFG) (16%)

- Concebir las especificaciones del proyecto que cubran las necesidades del sistema (40 horas).

- Diseño del software. (40 horas)

CONSTRUCCIÓN/PRUEBAS UNITARIAS (40%)

- Programación del software. (150 horas)
- Implementación del sistema completo. (50 horas)

PRUEBAS INTEGRADAS (20%)

- Evaluación de todo el sistema y corrección de errores. (100 horas)

DOCUMENTACIÓN MEMORIA / DEMO (24%)

- Redacción de la memoria del trabajo fin de grado. (100 horas)
- Demo de presentación (20 horas)

Total: 500 horas

Como se indica anteriormente, la distribución es aproximada y tras realizar todas las tareas, se realiza el diagrama del siguiente apartado mostrando gráficamente la duración por semanas de cada punto clave del proyecto. El total de horas al final del ejercicio asciende a 700 también de manera aproximada.

3.8 Diagrama temporal

DIAGRAMA TEMPORAL

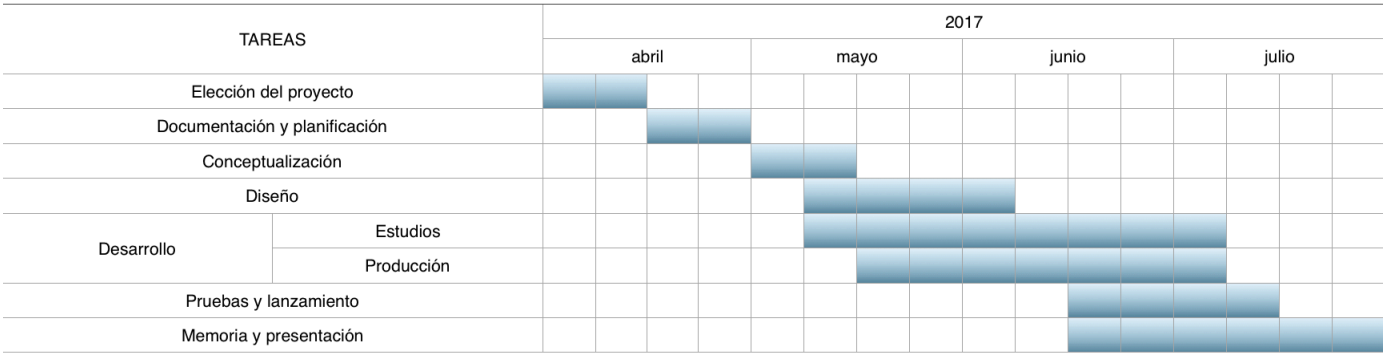

Tabla 1. Diagrama con las funciones realizadas por semanas.

En el mes de abril, se aprecia que las primeras dos semanas se dedican a la elección del tema para el trabajo de final de grado. En este periodo reflexiono sobre cuál de los diversos proyectos que tengo en mente llevar a cabo. Los dos que más me motivaban eran:

- Aplicación para la detección de impacto en vehículos estacionados.
- Aplicación capaz de detectar ultrasonidos.

Decidí elegir la segunda opción porque encontré la empresa donde desarrollarlo y adquirir mayores cualidades en un entorno de trabajo real.

Las siguientes dos semanas me incorporé a la empresa, conocí el funcionamiento interno de la misma y estuve investigando sobre el tema. Además, comencé a repasar los lenguajes de programación que se iban a utilizar y realicé un curso muy completo sobre Android para ponerme en situación de lo que iba a tener que utilizar.

Realmente es en el mes de mayo cuando comienzo a decidirle tiempo al inicio de la creación de la aplicación. Decido dedicarle aproximadamente dos semanas para la estructura que va a seguir la aplicación, para hacerla lo más sencilla e intuitiva posible para el usuario.

Al diseño le he dedicado un mes, para decidir el diseño final que tendrá, ya que le doy mucha importancia a la parte gráfica y los detalles que pueda tener para impactar visualmente.

La parte de desarrollo se divide en dos partes, con *estudio* me refiero a toda la investigación que he tenido que hacer y la información que he tenido que buscar para conseguir que funcionase cada requisito de la manera esperada. Con *producción* me refiero a escribir literalmente el código. Se puede observar que, desde el principio hasta el final, ambas tareas han ido prácticamente ligadas, y esto tiene sentido, ya que anteriormente nunca había realizado un proyecto parecido y debía documentarme para avanzar en cada cosa por pequeña que fuera.

El siguiente bloque, pruebas y lanzamiento, con *pruebas* me refiero al tiempo que se ha probado la aplicación con la mayoría de funcionalidades implementadas y cómo se comportaban con el código más complejo cada vez. Y con *lanzamiento* quiero decir la elección del repositorio/market donde subir la aplicación, así como la subida como tal de la apk de la aplicación, y el proceso de distribución de la misma.

Como es lógico, la memoria de este proyecto se decide empezarla casi al final. En mi caso, decido hacerlo de este modo para afianzar cada uno de los conceptos y métodos que se utilizan a lo largo de la creación de la aplicación y tener una visión más general del trabajo realizado.

Capítulo 4. Conceptualización y diseño

Este paso previo a la programación, es de vital importancia ya que nos ayuda a definir mejor la app tanto visualmente como por la parte de funcionalidad y acabado final de la mima.

Los pasos seguidos fueron:

1. Realización de un esquema con *BUBBL.US* (https://bubbl.us), para obtener un mapa mental de todas las funcionalidades de la app, y también para hacerse la idea del *Customer Journey* (la experiencia del cliente que se quiere alcanzar)

2. Una vez que se tiene un esquema general, se empieza a dibujar las pantallas con papel y lápiz, sin focalizarse en detalles como iconos, imágenes o textos.

3. Cuando se tiene el número de pantallas previamente dibujadas, se empieza a hacer *wireframes* con *SKETCH*.

4. Una vez ya tengamos definidos los *wireframes* anteriores, se realiza un prototipo para ver el resultado final de cómo debe ser la app. En este caso se exportan los diseños a *MarvelApp* y se les da vida a las pantallas.

4.1 Mapas mentales

Como primera idea se pensó en el siguiente mapa.

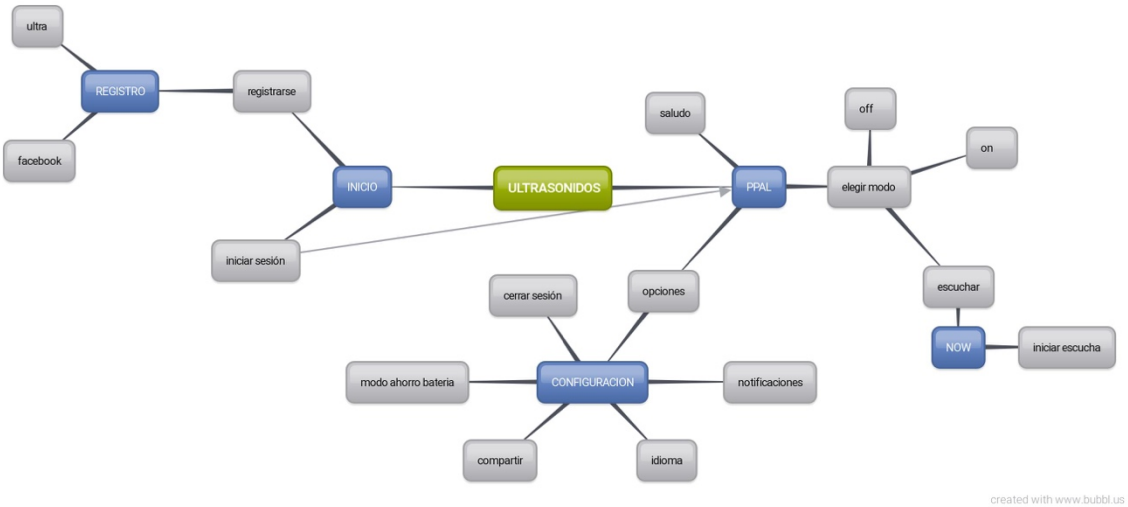

Figura 1. Mapa mental v1 de la estructura de la app.

Tras revisarlo con diseñadores y desarrolladores, se decidió realizar varios cambios, quedando el siguiente mapa.

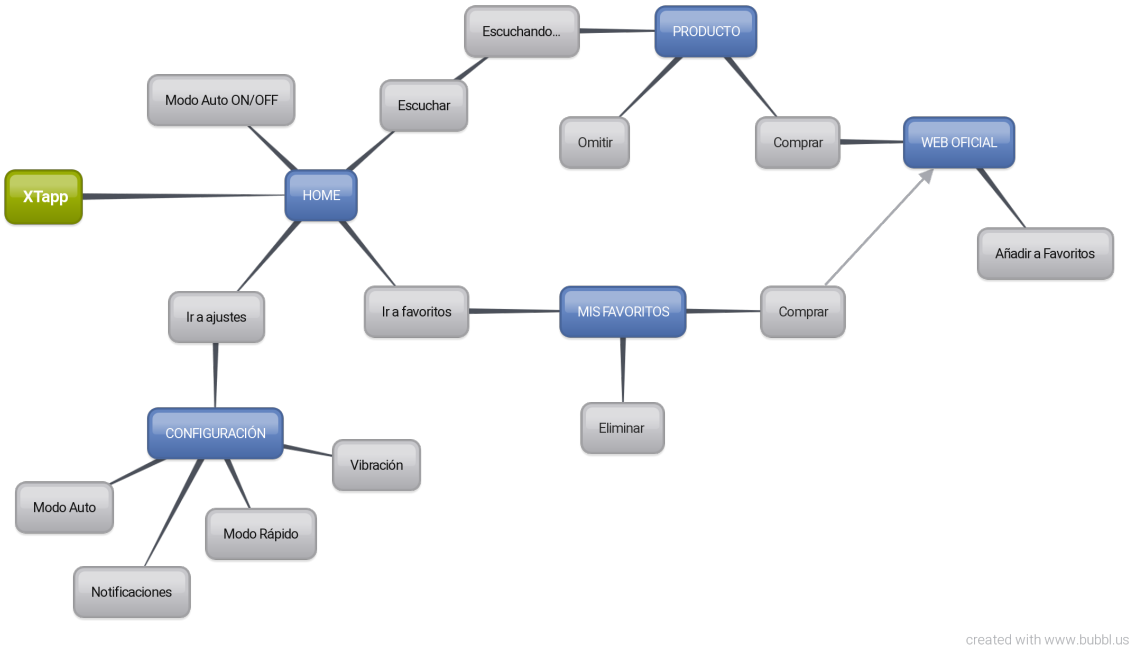

Figura 2. Mapa mental v2 con la estructura de la app.

Las diferencias entre ambas figuras, son realizadas para una mejor experiencia de usuario, para hacer la aplicación más intuitiva y más fácil de manejar. Por otro lado, también el mapa mental sufre cambios por motivos de las llamadas *guías de estilo* de Android.

Las *guías de estilo* son documentos donde se indica el formato y el diseño que deben tener por lo general las aplicaciones de Android.

Esto fue un reto para mí, porque soy usuario de iOS desde 2008, con lo cual, tuve que investigar y aprender cómo era el funcionamiento de las aplicaciones en Android, las partes de la pantalla, la función de cada botón, los gestos más comunes, donde colocar el menú y los ajustes, el botón *atrás,* etc.

Junto a la ayuda de diseñadores de aplicaciones de la empresa, se decidió la versión 2 como esquema general para estructurar todas las funcionalidades que tenía pensadas que debía tener la aplicación y tenerlo bien organizado.

4.1.1 Bocetos del diseño inicial

Siguiendo el consejo de los diseñadores se dibujó una serie de pantallas con sus estructuras generales y posibles logo de la aplicación.

En la siguiente figura se observan cuatro de las seis vistas pensadas para la visual de la aplicación.

Figura 3. Vistas iniciales que tendría la aplicación.

Se puede observar que la idea es tener un registro del usuario y posteriormente una pantalla principal donde tener diferentes modos junto a las últimas ofertas detectadas.

En la vista ofertas se aprecia una *Card* que dependiendo hacia donde la mueves aceptas o rechazas el producto junto a unos botones de compartir y más información.

En la siguiente imagen se encuentran las vistas restantes.

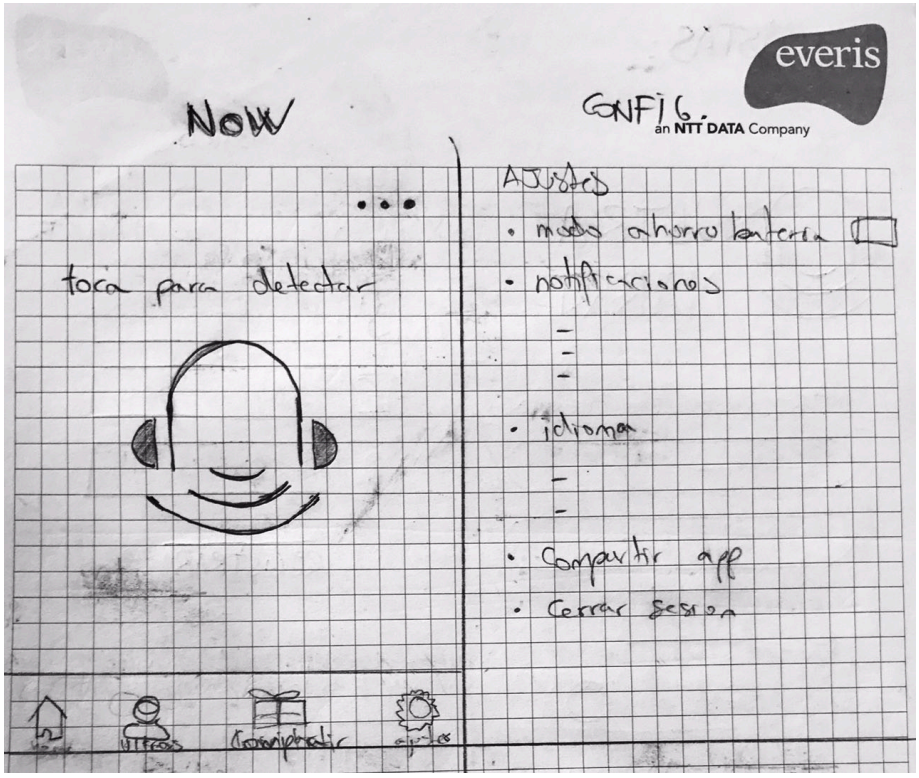

Figura 4. Vistas *Now y Ajustes* que tendría la aplicación.

Se observa una vista *Now* la cual al presionar en el centro de la pantalla empezaría la detección junto a un botón de menú y una barra inferior con diferentes opciones de navegación entre actividades.

En la vista de *Ajustes* se encuentras las opciones futuras que podría tener la aplicación.

En esta última figura se decide mostrar el aspecto inicial que iba a tener el logo o icono de la aplicación. Como se ven en las figuras anteriores, en un primer se pensó en llamar a la aplicación *UltraMarkt* por los ultrasonidos y la parte de marketing que tiene, por ese motivo en los siguientes logos se juega, tanto con el nombre como con las iniciales.

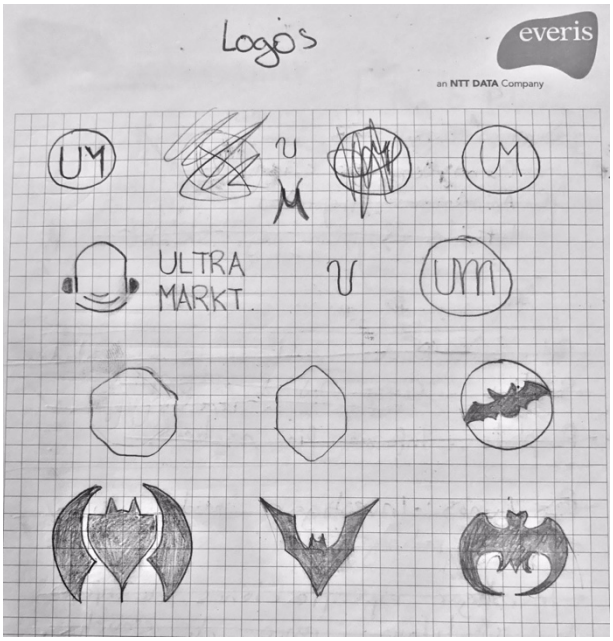

Figura 5. Diseño de logos/iconos iniciales de la aplicación.

Además de intentar darle sentido con el nombre y sus iníciales, se intenta jugar con el murciélago por dos motivos, es el animal que representa a mi ciudad de origen además de que concretamente los murciélagos utilizan los ultrasonidos para orientarse y gracias a que estas ondas se reflejan en su entorno tienen la manera de percibir objetos u obstáculos, resumiendo es si forma de ver.

4.1.2 Wireframes

Llegados a este punto la siguiente tarea consiste en digitalizar las ideas previamente expuestas. Para ello se crea un proyecto con *Sketch*. En este proyecto se deben dibujar todas las vistas pensadas e introducir mejoras siempre que sea posible.

En esta etapa se observa que si volvemos a los mapas mentales de las figuras 1 y 2 la estructura se muestra cambiante entre la versión del mapa uno y la versión del mapa dos.

Esto es normal, porque mientras avanza el proyecto se va revisando y comprobando cual es la mejor forma de implementar las cosas tanto por la parte de desarrollo como para la parte de experiencia al usuario.

En la siguiente figura se observa las vistas pintadas del proyecto mencionado en su primera versión.

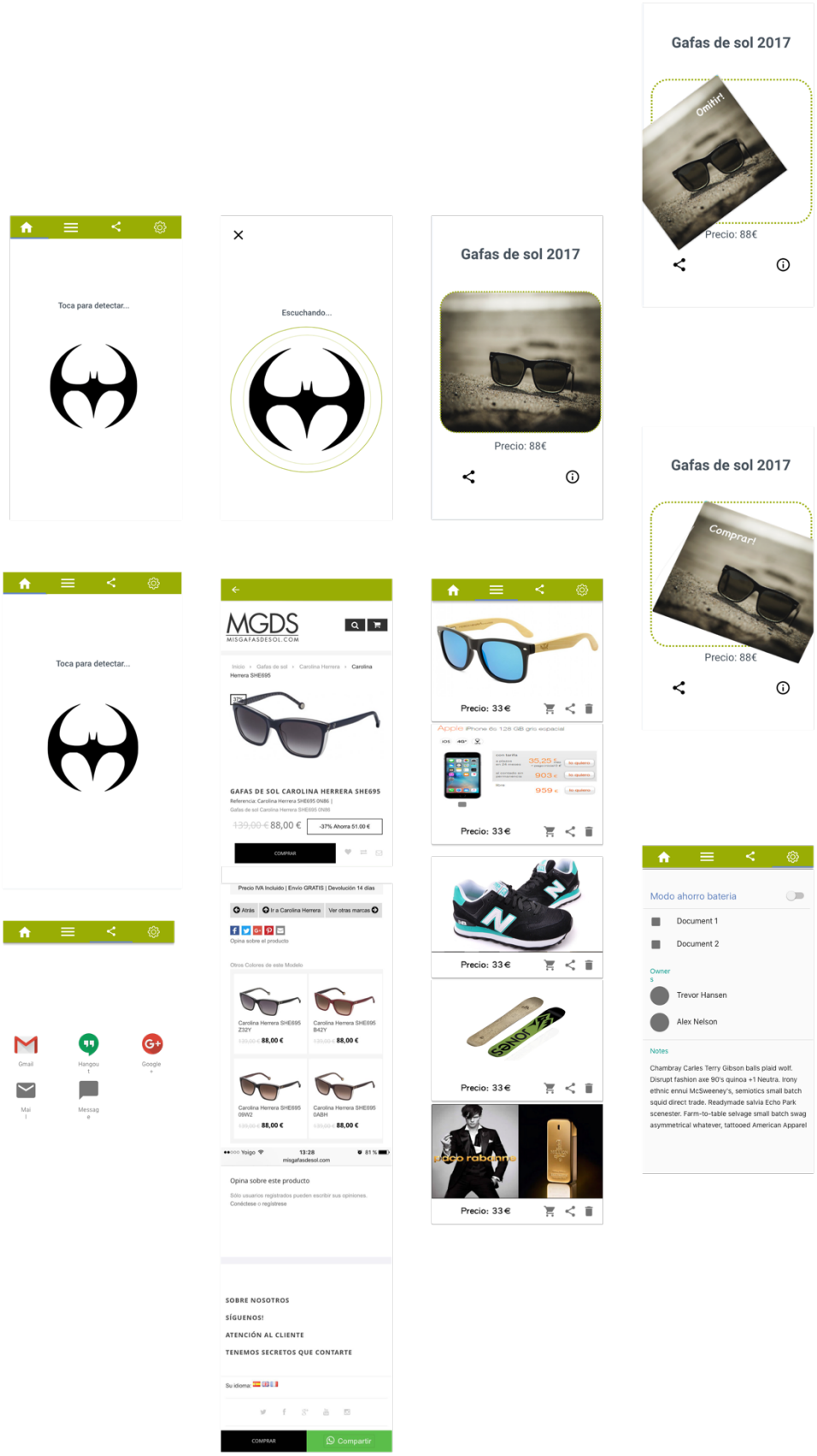

Figura 6. Versión 1 de las vistas de la aplicación con *Sketch.*

Una vez, la aplicación ya va cogiendo forma y mayor organización entre vistas, se decide cambiar el diseño para una experiencia mejor por parte del usuario, más llamativa, más moderna y sobre todo más cercana a los estilos de Android llamados *Material Desing.* Estos modelos son guías que te indican como tienes que ser los textos los iconos, donde debe ir el menú, como deben interactuar entre sí las vistas, etc.

Esta parte, para mí fue bastante tediosa y tuve que hacer un ejercicio de investigación importante, porque yo desde 2008 con la salida del iPhone 3G, no había utilizado un Smartphone con sistema operativo Android, es decir, nunca.

Así que desde la primera versión pintada hasta la final iba conociendo nuevas partes de las pantallas de Android, lo que se podía y no se podía utilizar, básicamente por la parte de la programación, porque Android para eso es bastante permisivo, permite cualquier estilo que se desee. Pero las *ActionBars* y las *NavgationBars* Android siempre las tiene muy presentes y hay que utilizarlas en muchos casos para ciertas acciones.

Como resultado de este ejercicio en las siguientes imágenes se muestran las vistas pintadas en formato digital las cuales formaran parte del prototipo explicado más adelante.

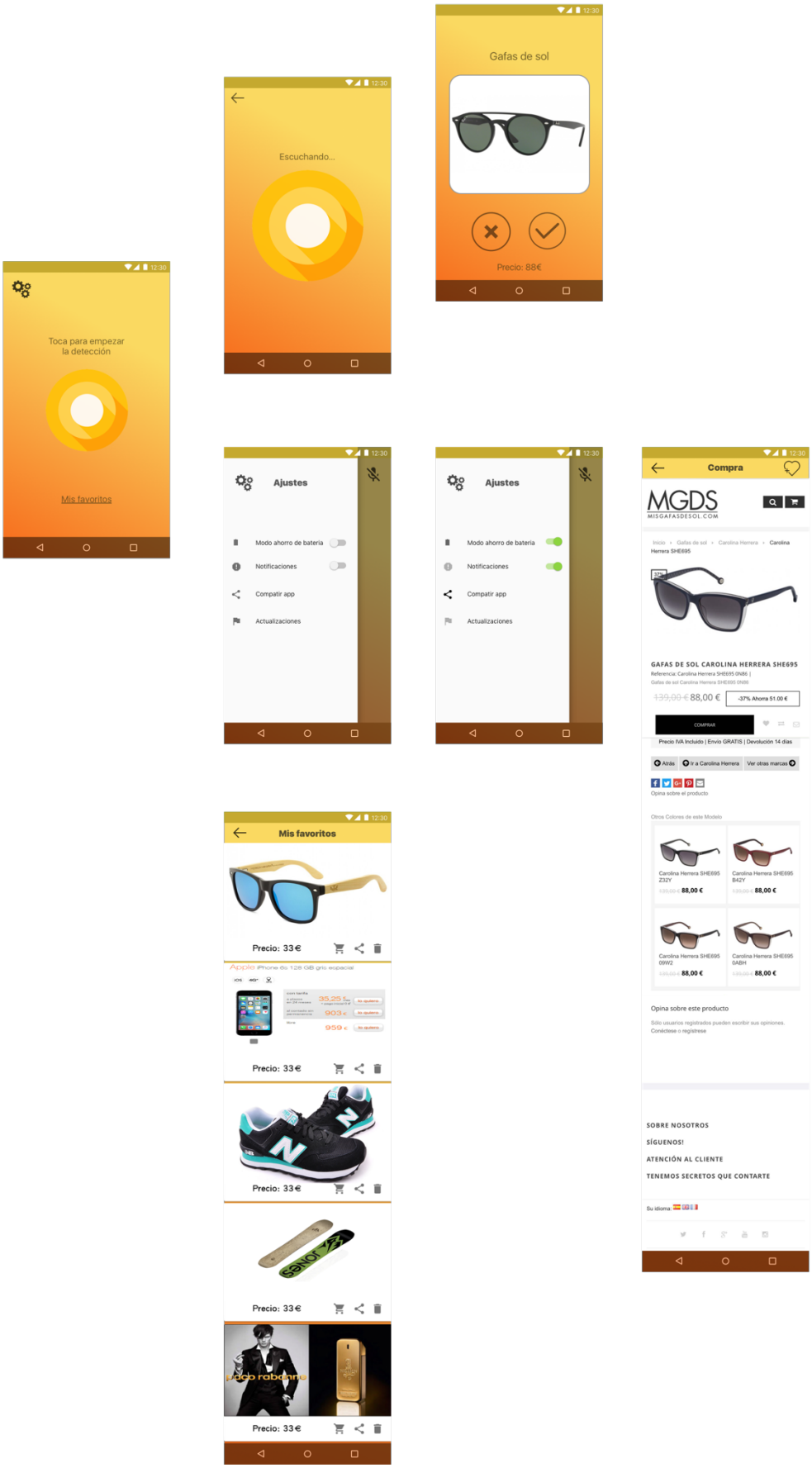

Figura 7. Versión 2 de las vistas de la aplicación. Actividades del modo auto apagado.

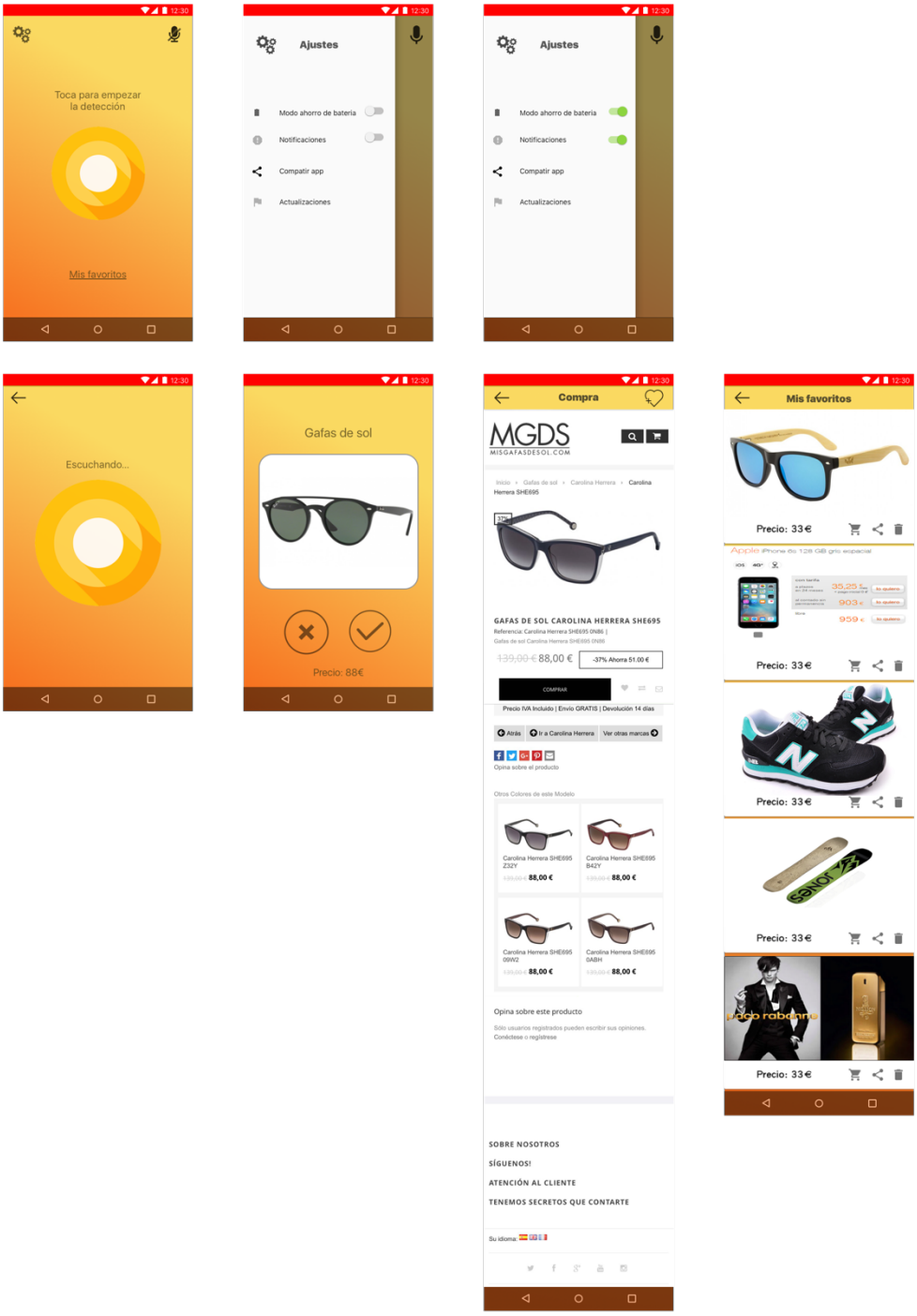

Figura 8. Versión 2 de las vistas pintadas de la aplicación, actividades con el modo auto activado.

En las figuras 7 y 8 se muestran las vistas tanto con el *Modo Auto* activado como en modo desconectado. La diferencia principal es que la *StatusBar* que es la barra superior del sistema operativo cambia de color para alertar al usuario de que esta función esta actividad. El icono del micrófono de la pantalla principal también cambia y para que el usuario sepa que tanto desde ahí como del menú que aparece por la parte de la izquierda se puede encender y apagar esta funcionalidad.

4.1.3 Prototipo

Es este apartado se muestran los proyectos creados con *MarvelApp* para diseñar el prototipo que en un proyecto real se enseñaría al cliente previamente a la contratación del servicio y este se hiciese una idea del resultado final.

Para llevar a cabo este proyecto se utilizan las vistas pintadas con *Sketch* de los *Wireframes* explicados en los puntos anteriores.

Con este paso, lo que se consigue es dotar de movimiento a las imágenes dando un efecto similar al que tendría la aplicación una vez estuviese implementada con código.

Es importante remarcar que el prototipo no tiene nada de código detrás, únicamente a cada vista se le indica que acción realizar cuando el usuario toque en ciertas partes de la pantalla.

Las acciones siempre serán navegar a otras imágenes insertada en el proyecto y se pueden personalizar los lugares donde el cliente puede interactuar y la transición entre imágenes que se desea para dotar de más realismo la demostración.

MarvelApp dispone de página web donde se puede realizar todo el prototipo.

En primer lugar, se exportan las vistas diseñadas con *Sketch* y posteriormente se suben a la web de *MarvelApp* para indicarlas las acciones explicadas.

En la siguiente imagen se muestra el primer prototipo con algunas de las vistas que se utilizan.

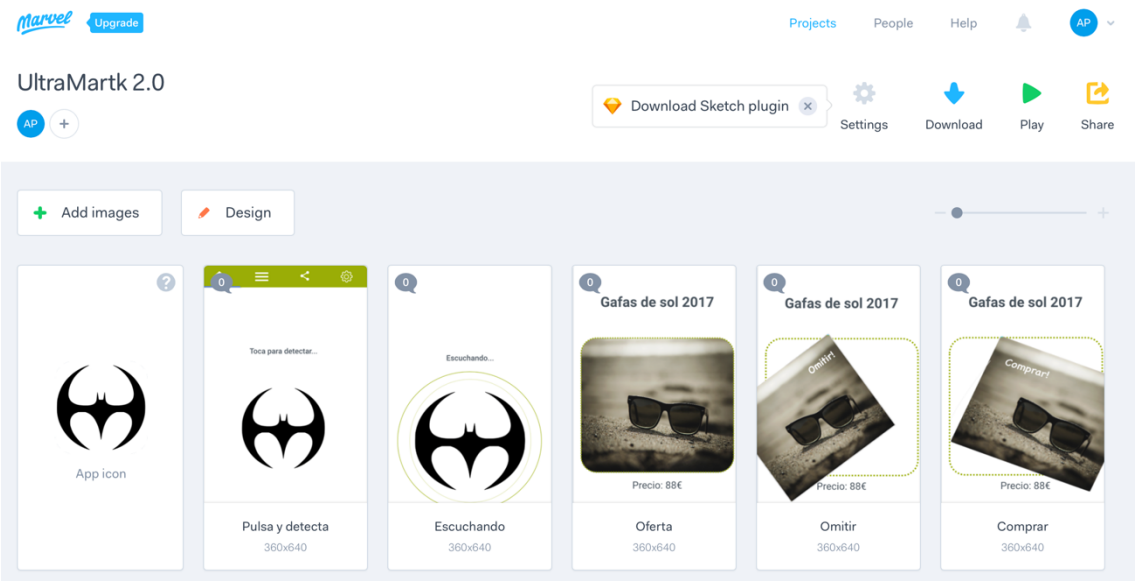

Figura 9. Primer prototipo de la aplicación con *MarvelApp*.

Como se observa se cargan las imágenes, y se exportan en el mismo tamaño que le indicamos que debía tener en *Sketch* y los mismos títulos para tener una mayor organización. En este caso no es necesario, pero en proyectos reales en este proyecto podemos encontrar cientos de imágenes y puede suponer una labor bastante complicada.
La primera imagen es donde subimos el diseño que tendrá el icono de la aplicación. Esto le da un valor añadido porque estos prototipos a través de un link se pueden visualizar en los smartphones independientemente sean Android o IOs, y pueden

quedar más reales creando un acceso directo al link en los móviles. Haciendo esto, MarvelApp utiliza la imagen llamada *App icon* como simulación del icono.
Cuando se accede a una de las imágenes se visualiza lo mostrado en la Figura 10.

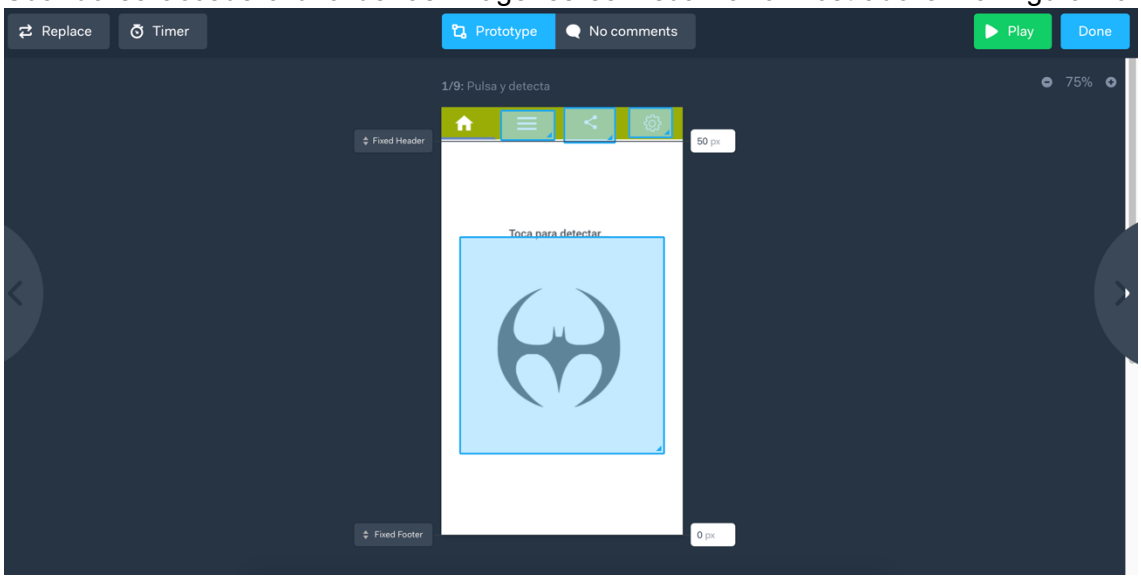

Figura 10. Muestra las opciones cuando accedemos a una de las imagenes.

Aquí se puede fijar los márgenes de la pantalla con *Fixed header* y con *Fixed Footer.* Con esto, podemos crea sensación de scroll en cuando el cliente este navegando por esta vista, siempre y cuando la imagen sea más larga que el tamaño del dispositivo.

Con el botón *Play* una vez tenemos todo configurado se visualiza como ha quedado la simulación.

Dicha simulación se consigue seleccionando diferentes partes de la imagen median recuadros dibujados por nosotros mismos para seleccionar toda la zona que realizar la transición.

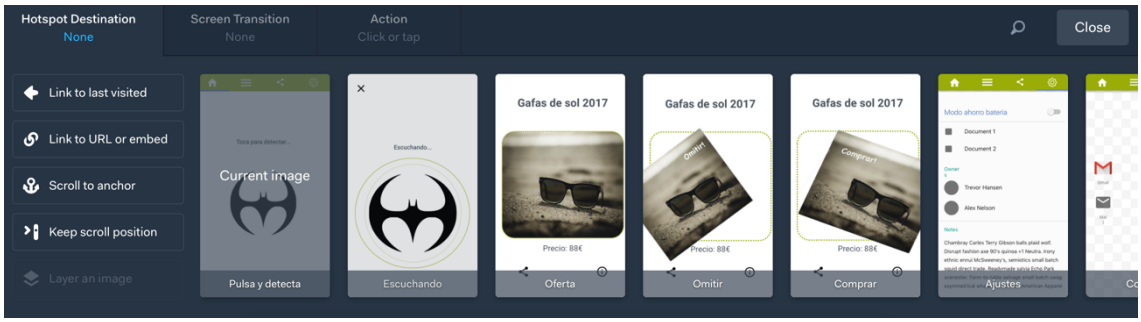

Figura 11. Configuración de acciones al crear un rectángulo en la imagen. Pestaña *Hotspot Destination.*

En la figura 11 se ve la pestaña de *Hotspot Destination* en la cual se elige la imagen a la que se navega al hacer click en la zona seleccionada.

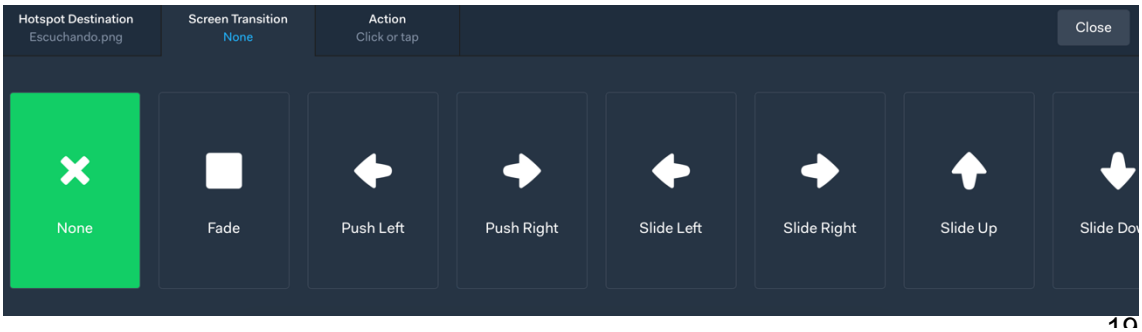

Figura 12. Configuración de acciones al crear un rectángulo en la imagen. Pestaña *Screen Transition*.

En la figura 12 se muestra la pestaña de *Screen transition.* Aquí se le indica que efecto se utiliza cuando se lleva a cabo la transición.

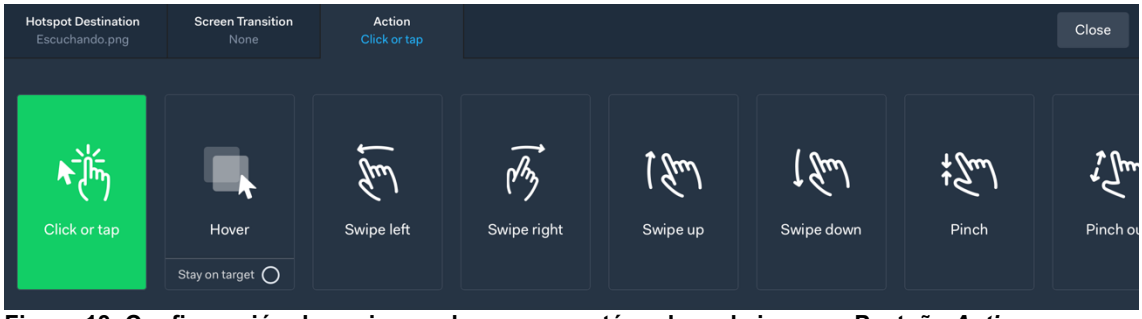

Figura 13. Configuración de acciones al crear un rectángulo en la imagen. Pestaña *Action*.

En esta figura 13, se visualiza los tipos de acciones que puede realizar el cliente sobre la pantalla y cuando se realice el seleccionado la transición se hará efectiva.

Una vez explicado el funcionamiento de esta herramienta, es necesario explicar que las imágenes son del prototipo versión 1. Por cada, *wireframe* se realizó un prototipo para tener más claro cuál sería el resultado final.

En la siguiente figura se muestran los dos proyectos que albergan los dos prototipos.

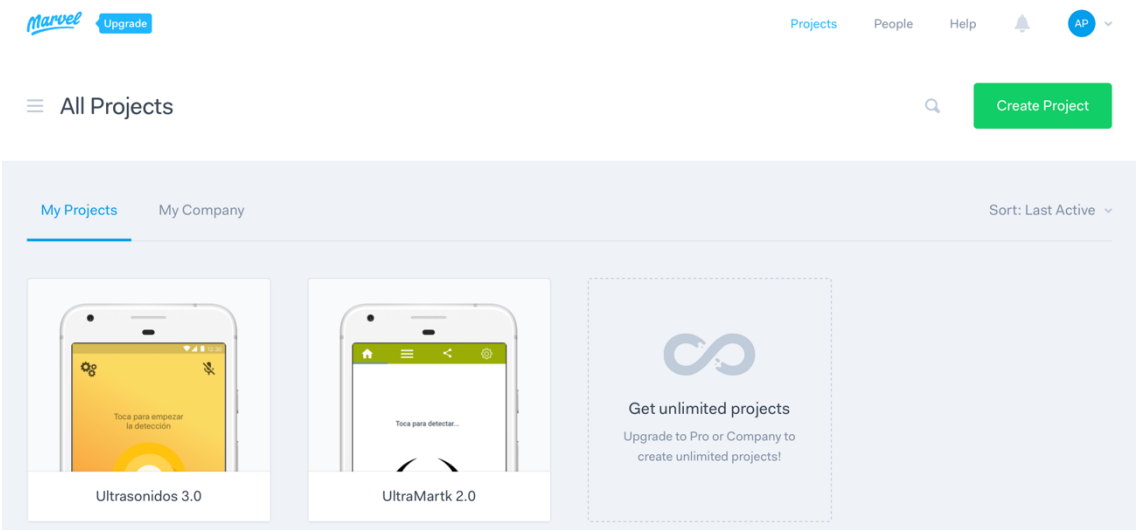

Figura 14. Prototipos creados en *MarvelApp*.

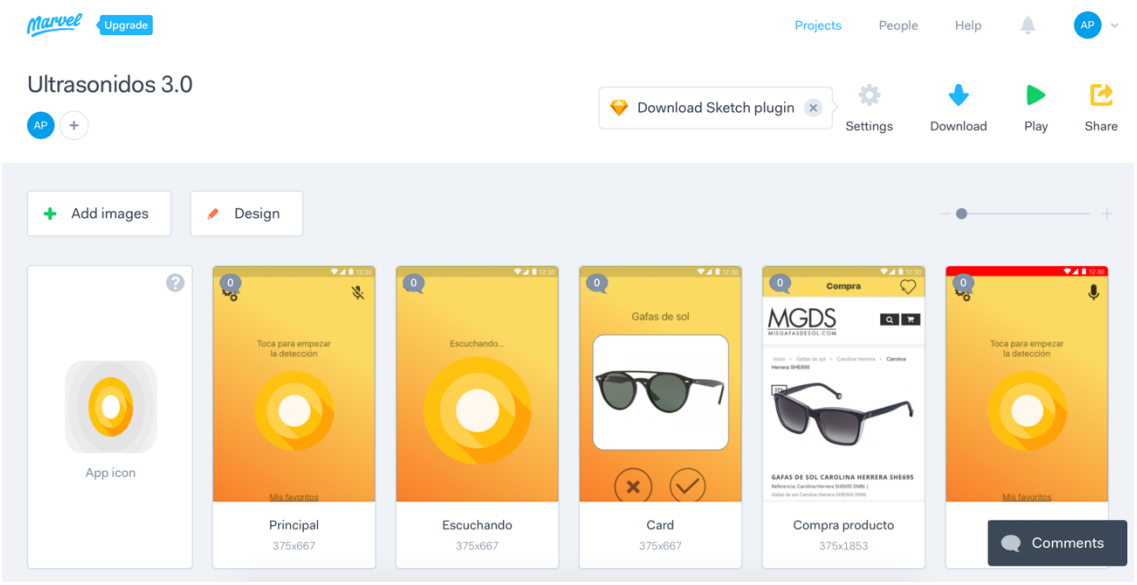

Figura 15. Imágenes del prototipo final con *MarvelApp*.

En la figura 15 se muestran algunas de las imágenes cargadas en *MarvelApp* para la creación del prototipo final.

Una vez el prototipo está terminado, este se puede visualizar e interactuar con él tanto en la web de *MarvelApp* como en cualquier dispositivo móvil.

Los links generados para estos dos proyectos son:

- Prototipo inicial: https://marvelapp.com/449g59d
- Prototipo final: https://marvelapp.com/37jb8e9

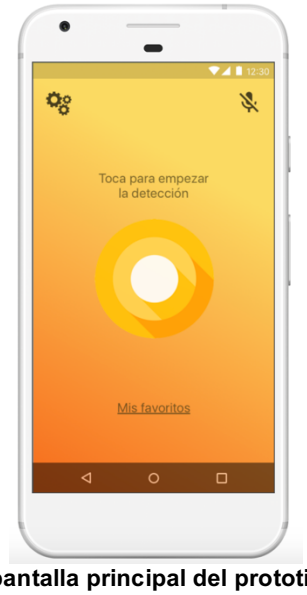

Figura 16. Vista de la pantalla principal del prototipo en la versión web de *MarvelApp*.

Capítulo 5. Desarrollo

Llegados a este punto, teniendo la idea clara de diseño y conceptualizando, es cuando se empieza realmente a implementar la aplicación.

En este caso, se utiliza *Android Studio* como entorno de desarrollo. La versión es la 2.3.2.

Lo primero que se hace es instalar el software y configurarlo debidamente para evitarse fallos futuros.

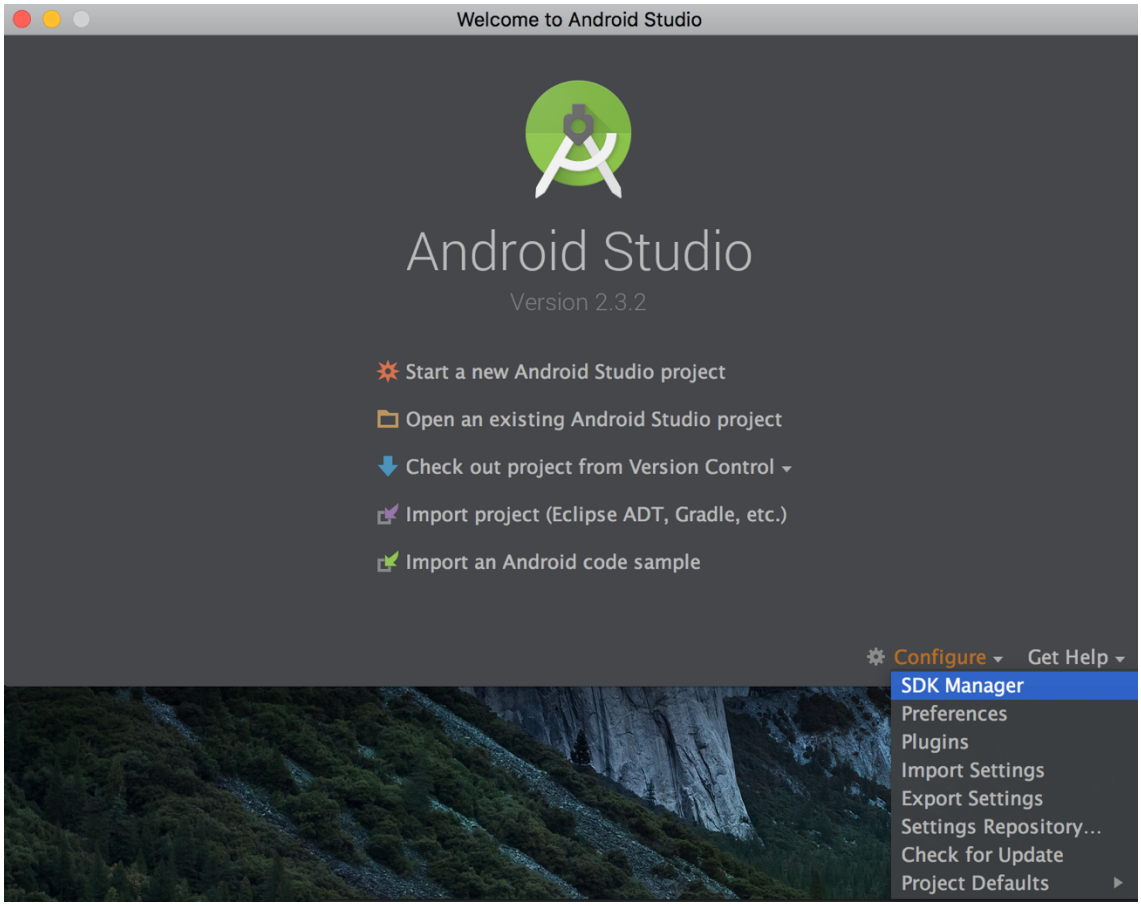

Figura 17. Selección de pestaña *SDK Manager* para configurar el SDK.

En la figura 17 se ve la pantalla principal y se indica dónde encontrar la opción para la configuración del SDK. Al seleccionarlo se abre la siguiente pantalla.

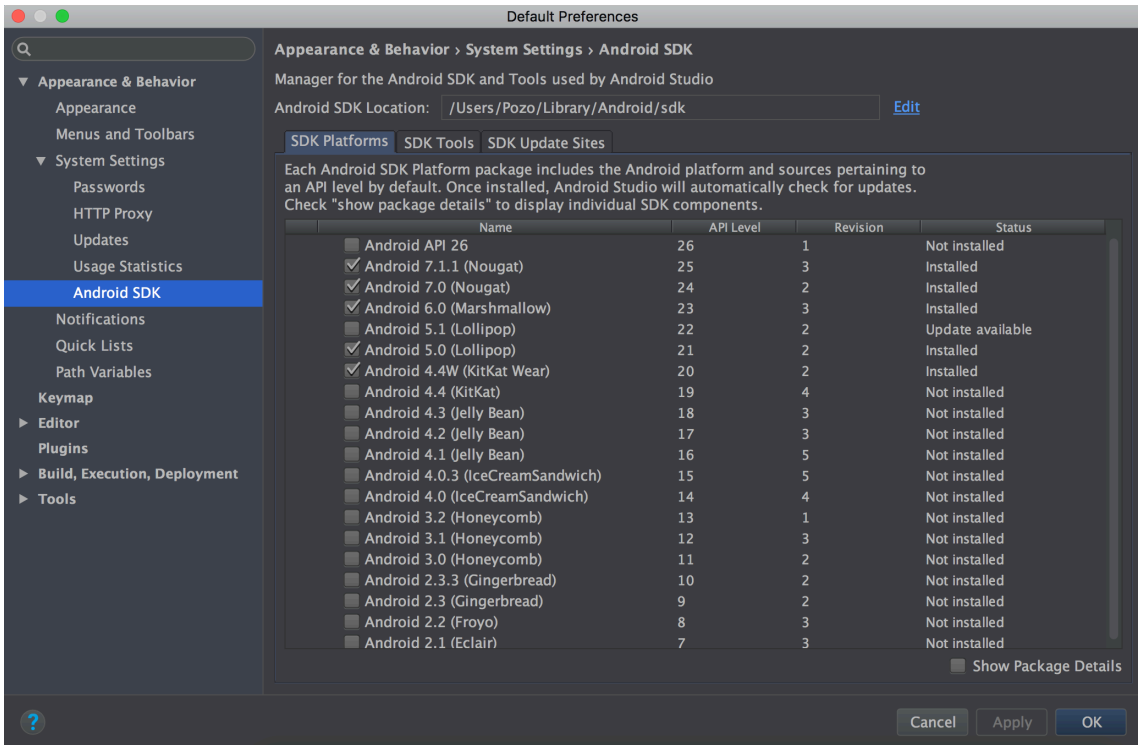

Figura 18. Pantalla de configuración de las versiones del SDK de Android.

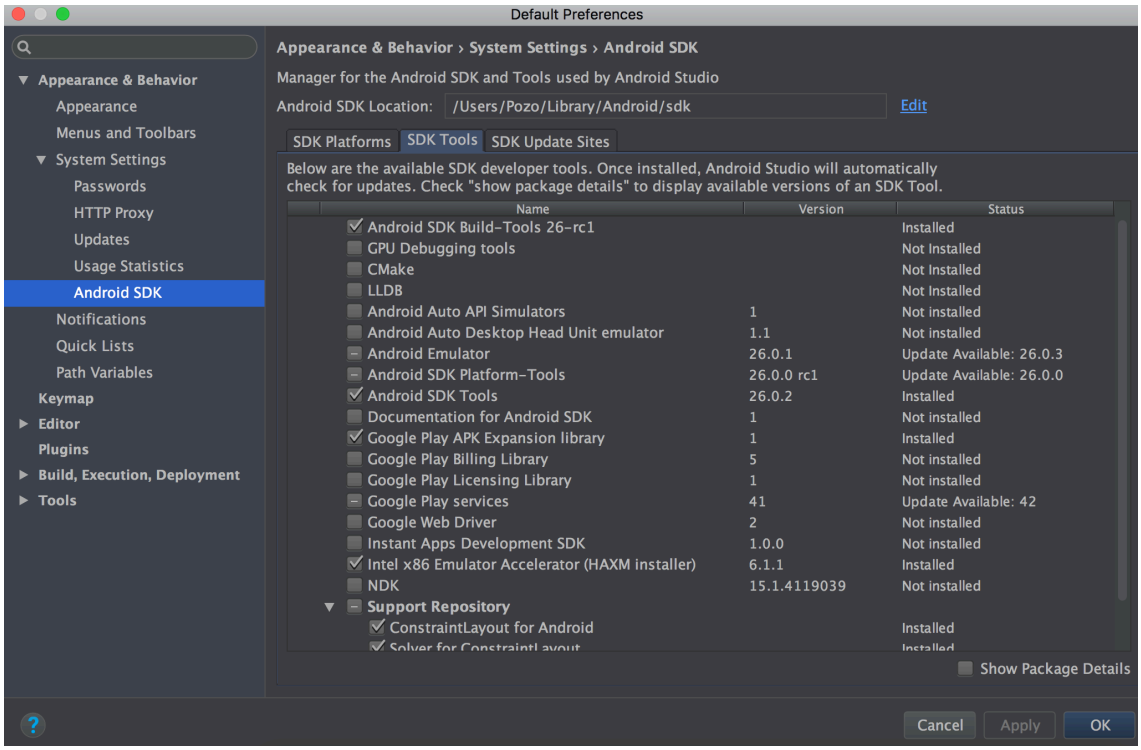

Figura 19. Pantalla de configuración de las herramientas del SDK de Android.

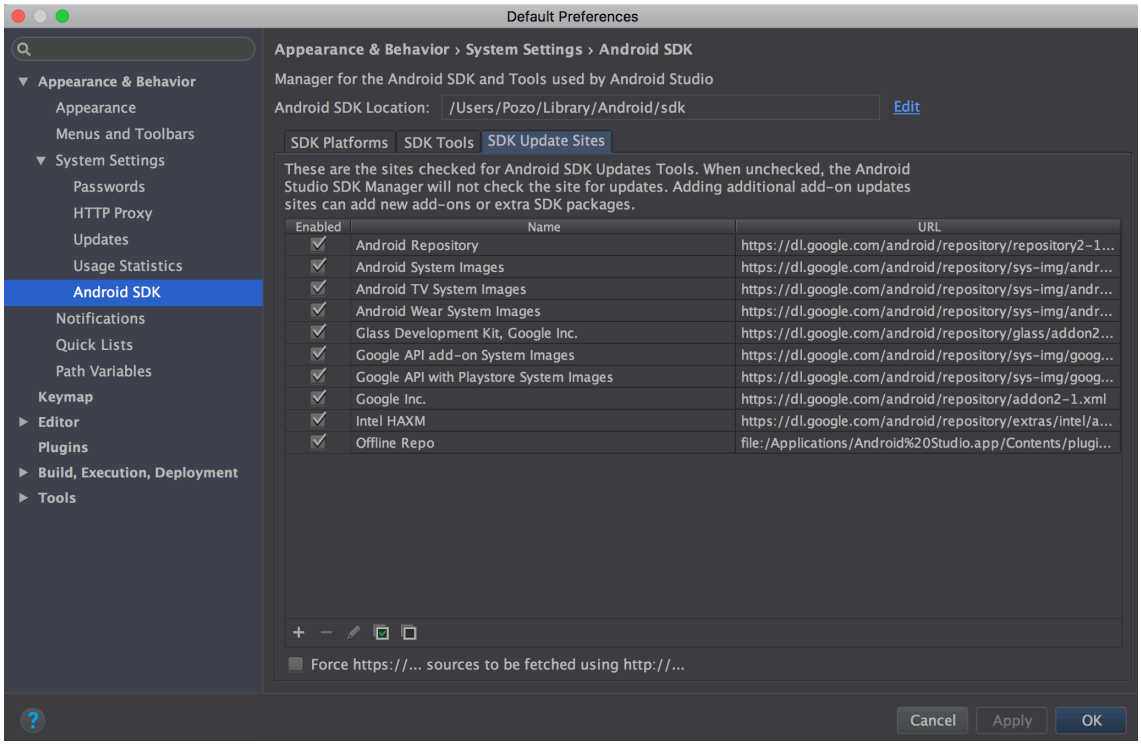

Figura 20. Configuración de componentes adicionales del SDK de Android.

En las figuras 18, 19 y 20 se muestras las tres pestañas que se encuentran dentro de la configuración del SKD de Android.

La figura 18 se centra en los componentes de las APIs que se necesitan para compilar posteriormente el proyecto dependiendo de las versiones para las cuales la aplicación funcionará.

La figura 19 muestra las herramientas necesarias para llevar a cabo el desarrollo de la aplicación.

La figura 20 alberga las posibles instalaciones que se pueden necesitar, siendo componentes adicionales como pueden ser los servicios de *Google Play*, preparar el entorno para desarrollar aplicaciones para dispositivos *wear, tv,* etc.

Las APIs, herramientas y componentes adicionales que se muestran en las figuras anteriores son las que han sido necesarias para desarrollar el proyecto del cual habla esta memoria.

Una vez se tiene la configuración realizada se crea un proyecto nuevo para comenzar a implementar el código necesario para la aplicación.

Las siguientes imágenes muestran los pasos a seguir para crear el proyecto en *Android Studio.*

Figura 21. Primer paso para la creación de un proyecto nuevo en *Android Studio.*

Al abrir *Android Studio* se abre la pantalla de la figura 21 y se debe clickar en la opción seleccionada *Start a new Android Studio project.*

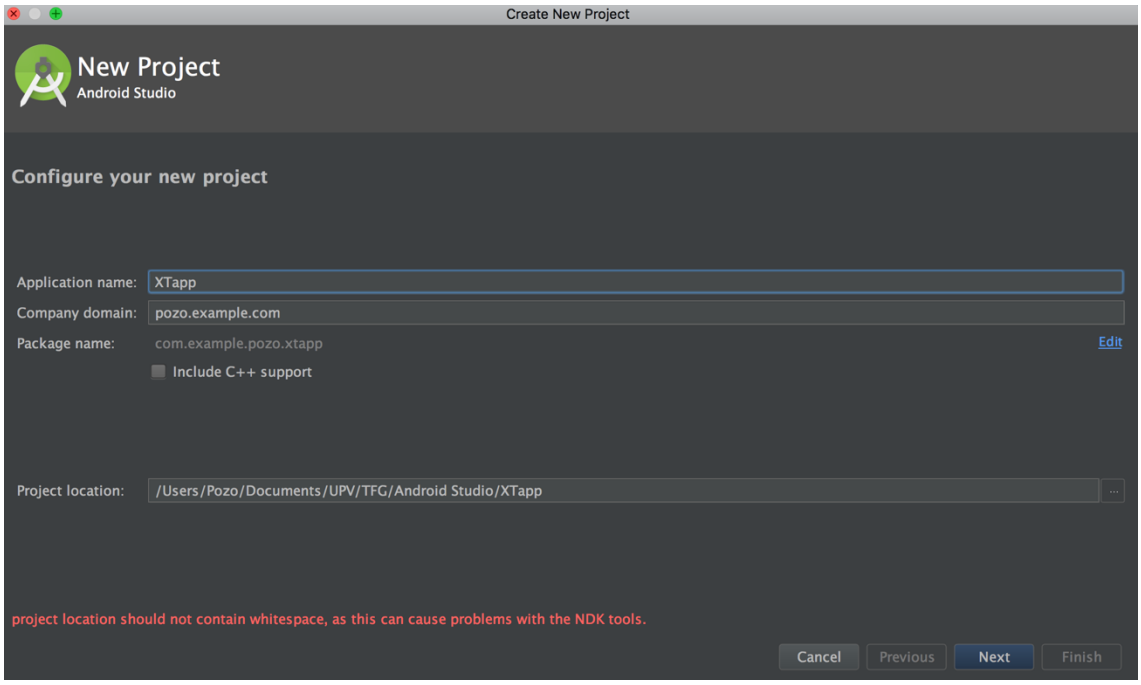

Figura 22. Pantalla asignación de nombre del proyecto y directorio.

En esta pantalla *Android Studio* permite seleccionar el nombre del proyecto que se va a utilizar y el directorio donde se guardará el proyecto junto a todos los archivos

necesarios para el desarrollo del mismo como, por ejemplo, clases, archivos JSON, librerías, etc.

Al presionar el botón *Next* se pasa a la siguiente pantalla.

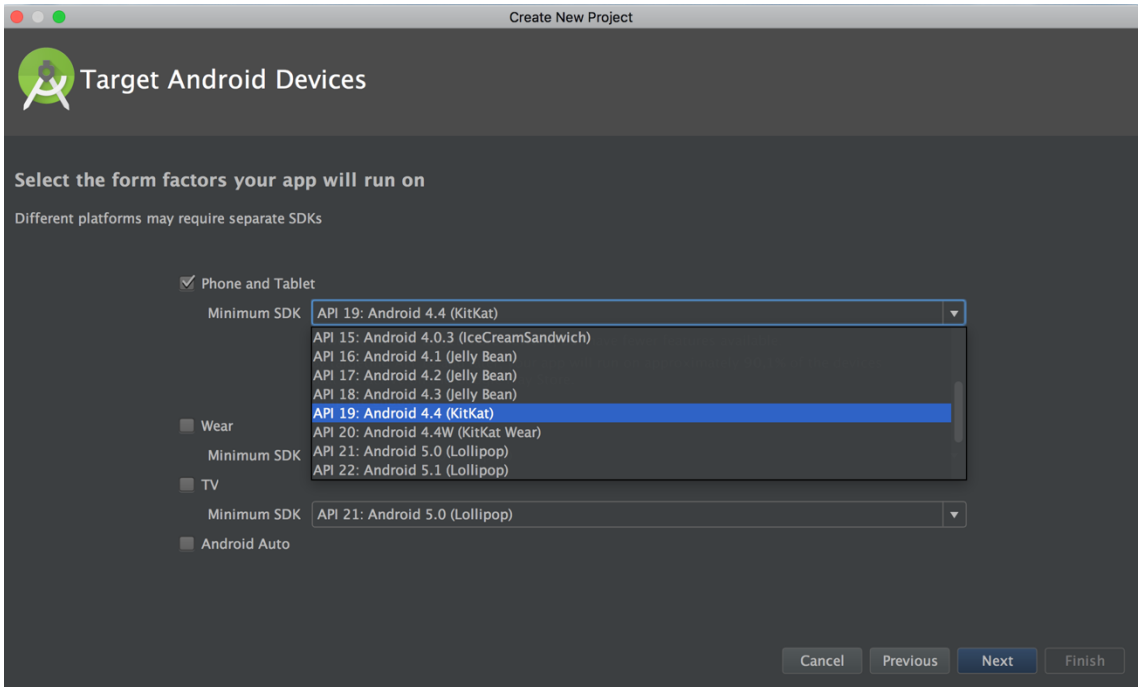

Figura 23. Pantalla para la selección de la API mínima que soportará la aplicación.

Aquí se decide para que versiones se va a crear la aplicación. *Android Studio* exige que se la API mínima para la cual funcionará el código.

Esta decisión es complicada y hay que tener un equilibrio. Siempre se quiere que las aplicaciones funcionen para el mayor número de versiones, pero esto puede provocar errores que se traducen en peor funcionamiento en general de las apps.

En el caso de este proyecto se decide que la API mínima es la 19, versión KitKat, por lo general es la más recomendable en estos momentos.

Para hacerse una idea del número que dispositivos que hay en el mundo operando en el rango de esa versión mínima a la más actual (API 25 versión 7.1.1 *Nougat*) en la misma pantalla de la figura 23 se encuentra un link en color azul donde pone *Help me choose.*

Al hacer click en él se abre la siguiente ventana.

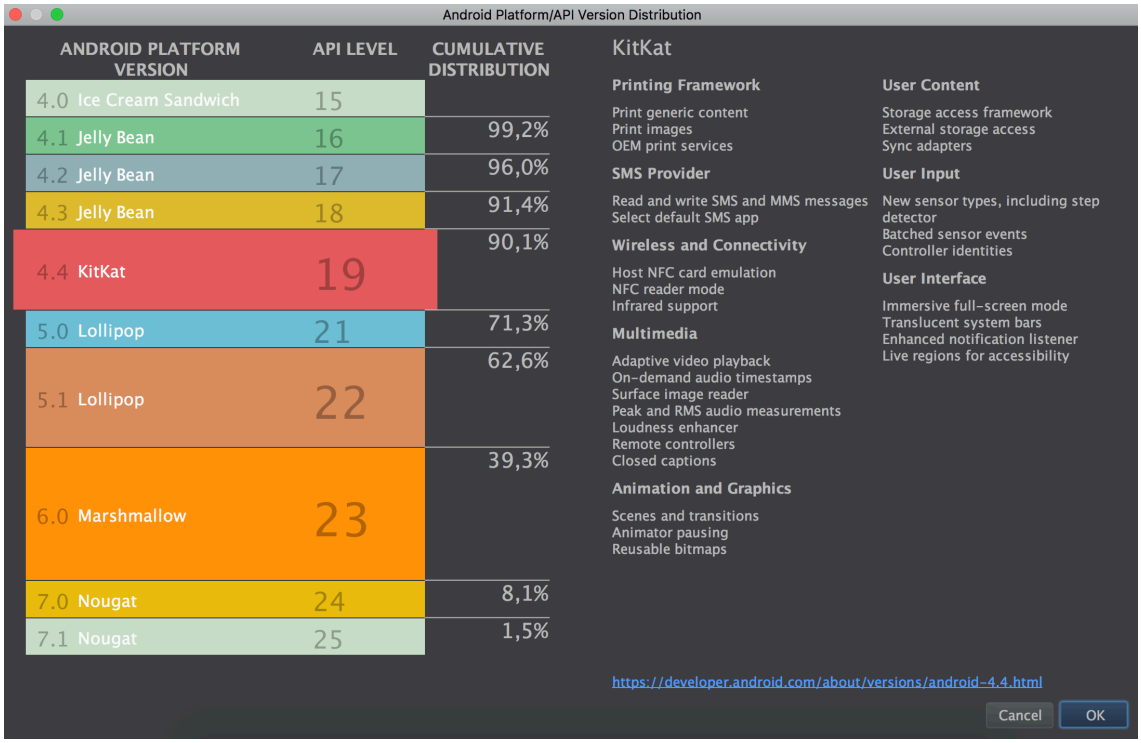

Figura 24. Estadística de las versiones utilizadas en la actualidad.

La figura 24 ofrece unos datos estadísticos proporcionados por *Android* donde se aprecia de manera simple y rápida las versiones que utilizan los dispositivos con un sistema operativo *Android* a nivel mundial.

Eligiendo la versión *KitKat* la aplicación que se va a desarrollar se puede utilizar en el 90,1% de los dispositivos que están utilizándose actualmente.

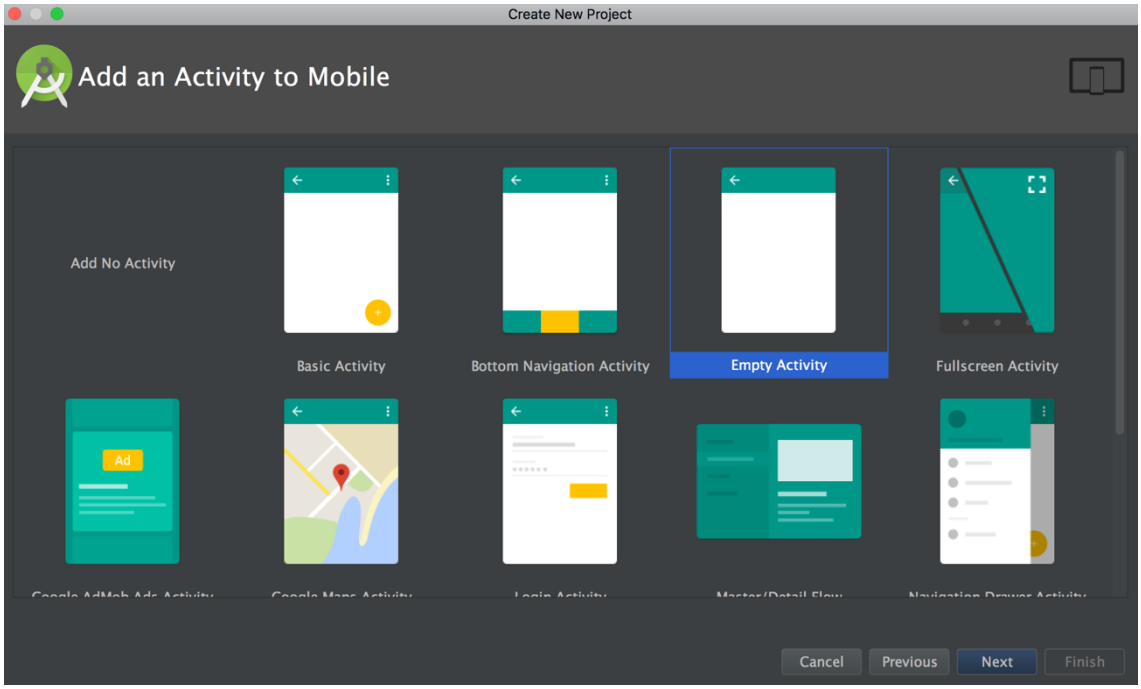

Figura 25. Tipo de actividad se generará por defecto al crear el proyecto.

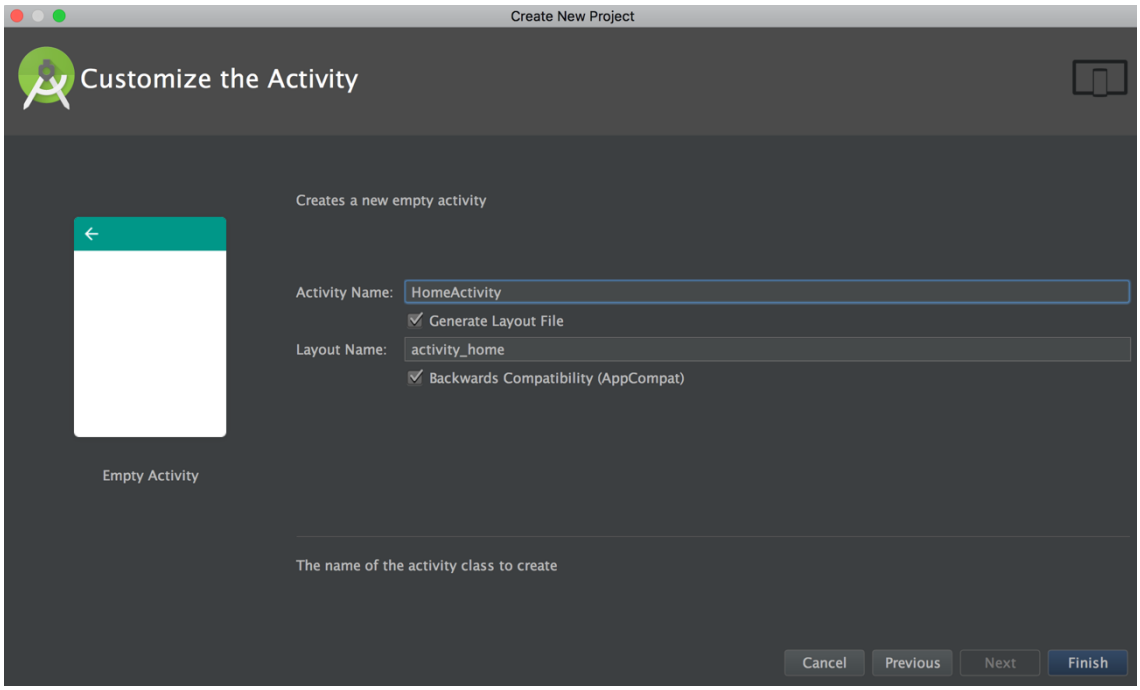

Figura 26. Asignación de nombre a la Actividad principal

En las figuras 25 y 26, se muestran los últimos pasos para la creación del proyecto. En la figura 25 se elige el tipo de actividad que se generará por defecto al crear el proyecto y en la figura 26 se le asignan nombre, en este caso *HomeActiviy.* Al presionar el botón *Finish* se termina la configuración del nuevo proyecto.

Android Studio construye el proyecto y aparece la siguiente pantalla.

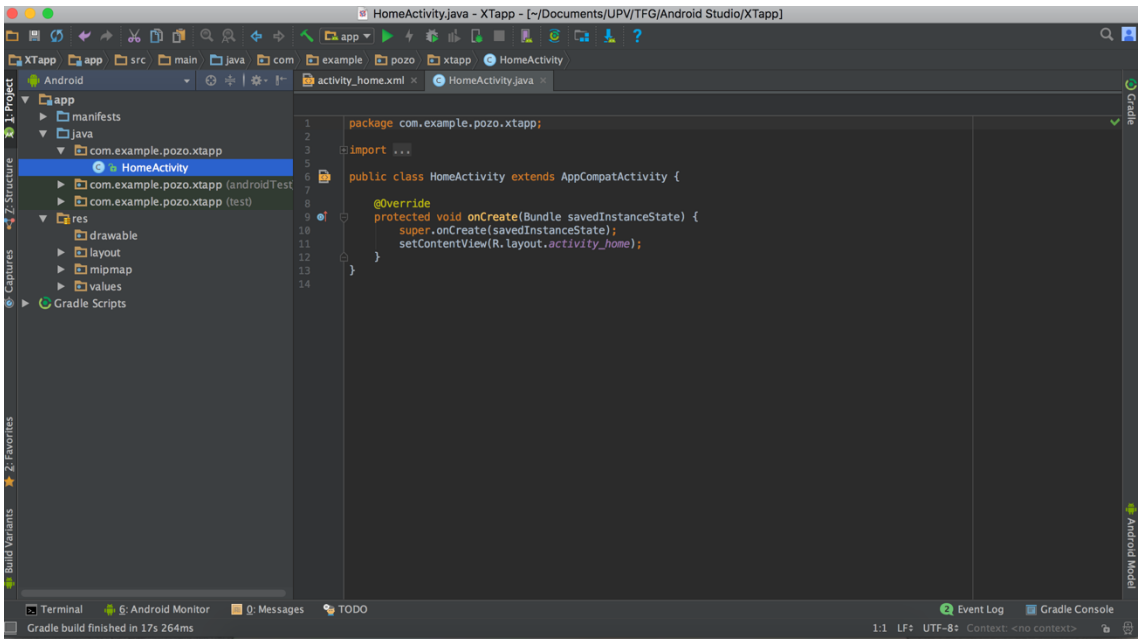

Figura 27. Primera imagen tras crear el proyecto en *Android Studio.*

Una vez se crea el proyecto en *Android Studio,* muestra la configuración realizada en las figuras anteriores y está listo para empezar escribir código para empezar a desarrollar la aplicación.

Cuando se tiene realizado esto, se explica el código utilizado para el funcionamiento de la aplicación en cuestión. En los siguientes apartados se detalla y explica de manera narrativa cuales son las funciones de la app y como se consiguen llevar a cabo y hacerlas funcionar.

5.1 Análisis de requisitos

La aplicación cuenta con varias funcionalidades, se va a proceder a enumerarlas según el orden de implementación de las mismas.

- Detección de ultrasonido mediante botón escuchar.

Esta funcionalidad consiste en presionar un el botón principal de la aplicación para iniciar el proceso de escucha. Tras iniciar este proceso, la aplicación detecta el ultrasonido y te muestra por pantalla la información asociada a ese audio, como puede ser la imagen del producto, su precio, el modelo.

- Compra del producto.

La compra del producto se realiza a través de la web oficial de cada marca.
Se puede acceder a esta funcionalidad inmediatamente después de la detección o mediante la batería de productos alojada en los favoritos de cada usuario.

- Modo Automático.

Este requisito está en constante escucha de XTAudioBeacons al dispositivo para posteriormente notificar al usuario de las detecciones realizadas.
Este modo realiza la detección a través de un servicio de Android, en segundo plano y sin necesidad de tener la aplicación en primer plano, pero si abierta en background.

- Mis Favoritos.

Es una funcionalidad para darle mayor uso a la aplicación, se pensó implementarla para que los usuarios tengan más opciones de poder realizar la compra ya que en un primer momento, si no se realizaba la detección en el momento no se podía llegar a la página oficial de la oferta en concreto. Con esta vista, se consigue dar la opción al usuario de, además de por omitir o realizar la compra, poder guardar la oferta detectada para acceder a ella posteriormente. Para implementar con éxito esta funcionalidad se utiliza una base de datos insertada en la misma aplicación, de cada usuario.

5.1.1 Requisitos funcionales

Para un mayor entendimiento, se dividen en las diferentes funcionalidades previamente explicadas.

- Detección de ultrasonido mediante botón escuchar.

La actividad principal consiste en el botón escuchar, el cual empieza a detectar al presionarlo. Esto se consigue implementando la clase *ListeningActivity.java*.
Esta clase, aparte de tener diferentes métodos para que funcione como se espera, hereda de la clase *XTUltrasonicsActiviy.* Esta clase pertenece a una librería de terceros llamada *XTAudioBeacons.* Esta librería se puede encontrar en *Github*. Cuenta con una licencia MIT con lo cual no hay problema de utilización.

Dicha librería, recibe el audio que se registra con el micrófono del Smartphone. Posteriormente le aplica un filtro paso-alto con una frecuencia de corte de 16 kHz. Cuando se eliminan las frecuencias que están por debajo de dicho umbral, se realiza una Trasformada de Fourier Rápida (FFT) para obtener el espectro frecuencial del XTAudioBeacon. Con esta información se diferencia un audio de otro.

En la figura 28 se muestra de manera gráfica el sonido que registrará el micrófono y el filtrado para quedarse únicamente con las frecuencias donde se encuentra el XTAudioBeacon.

Volviendo a la clase *ListeningActivity.java,* por un lado, se implementa un método llamada *alert* donde se indica la acción que debe realizarse cuando se complete la detección. Es decir, llevará al usuario a la vista del producto y vibrará además de emitir un sonido. Por otra parte, mediante un switch se le asigna a cada una de estas huellas un objeto de tipo producto mediante la variable *pos*, el cual tiene como atributos el nombre de la compañía, el modelo del producto anunciado, el precio del mismo, una imagen y la URL oficial donde se realiza la compra.

Para conseguir guardar tantos objetos producto como XTAudioBeacons se tienen, se crea un array dentro del archivo *prodcuts*.json.

Este archivo tiene que asociarse a la parte lógica del código. Esto se consigue creando una clase java llamada *Producto.java*, en la cual se indican diferentes métodos para poder consultar o modificar los parámetros antes explicado, llamados atributos de cada objeto producto.

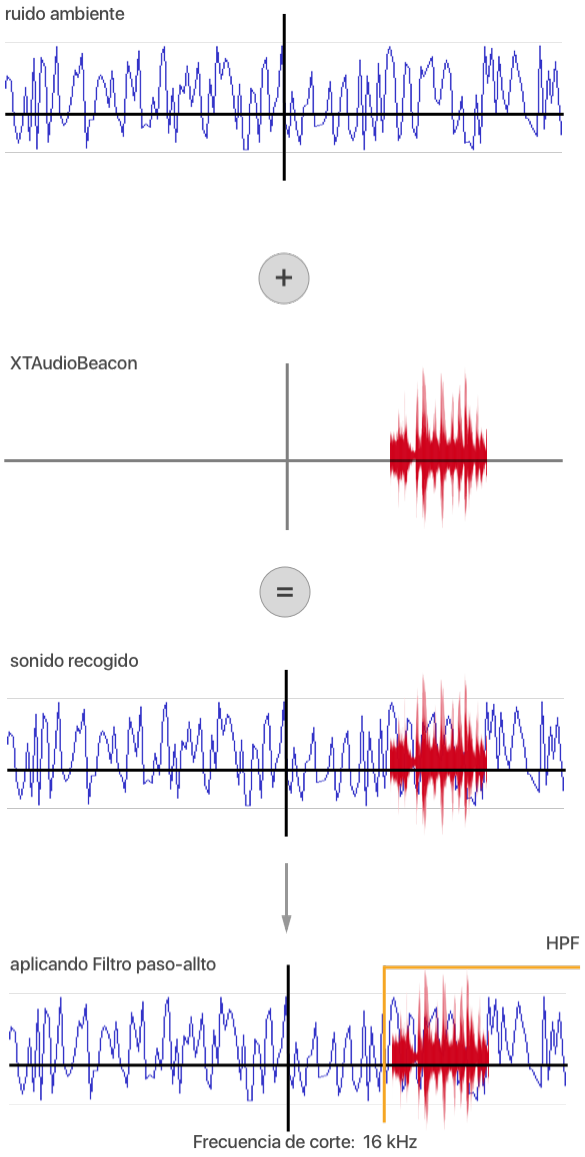

Figura 28. Pasos de la detección de los XTAudioBeacons (Ultrasonidos).

- Compra del producto.

Esta parte se soluciona de manera sencilla, redirigiendo al usuario y haciendo que sea la página oficial, la que gestión la compra.
Para visualízala dentro de la aplicación se utiliza un *WebViews*.

- Modo Automático.

Para darle una mayor utilidad a la aplicación, se decide implementar un modo automático que te permita recibir las ofertas sin necesidad de tener la aplicación abierta.
Para llevar a cabo esta función, se implementa un servicio de Android llamado *CaptureAudioService*.

Funcionando en segundo plano, la aplicación está en constante escucha y el servicio se encarga de mandar las escuchas al método de la librería externa que se encarga de procesar el audio. Posteriormente el funcionamiento es repetir la funcionalidad principal, detectar el ultrasonido, esta vez, sin necesidad del botón escuchar.

- Mis Favoritos.

Función que se traduce en una vista de Android, a la cual puedes acceder desde la pantalla inicial *HOME* a través de un *TextView* clickable.
En esta vista se encuentran los productos añadidos por el usuario desde el botón *Añadir a favoritos* implementado en la *ActionBar* de la *WebViewActivity*, actividad donde se realiza la compra.
Para llevar a cabo dicha función se utiliza una base de datos alojada en la misma aplicación. La tecnología que se utiliza es SQLite y el flujo que siguen los datos es el siguiente.
Como se tiene el archivo de tipo JSON llamado *products.json* alojado en la carpeta *assets*, ya se cuenta con todos los productos que pueden existir. Una vez estos productos pasan al código como objetos del tipo *Produc*t con la clase *Product.java* y cargando previamente el JSON con la clase *UTILS.java* se pasa el objeto *producto,* al cual se está accediendo cuando se realiza la detección, a la actividad WebViewActivity. Una vez se tiene el objeto en esta actividad, si el usuario decide guardar para pensar si quiere realizar la compra, debe presionar el botón *Añadir a favoritos,* situado en la *Toolbar* de la vista actual (*WebViewActivity*).
Con esto lo que empieza a procesar el código es:

- Primero recupera el objeto de tipo Product con el que se está trabajando.

- Se instancia la base de datos creada previamente en la clase *SQLiteFAvs.java* con *SQLite* llamada *favs.*

- Y se utiliza un método llamado *addProduct* de la clase *SQLiteFAvs.java* el cual, además de añadir el objeto *producto* en *Mis Favoritos* con todos sus campos (Empresa, URL de la imagen, precio del producto, URL de la página web oficial de compra del producto, modelo del producto, identificador), te devuelve una variable de tipo *entero* llamada *id* que funciona de la siguiente manera.

En el método llamado *addProduct,* la variable *id* se inicializa con el valor -1. Este método añade, como se ha explicado, el producto en una fila nueva siempre y cuando este registro no se haya realizado previamente. Si el registro se lleva a cabo, a *id* se le asigna el número de la fila donde se realiza el registro. Si ese producto ya estaba guardado el registro no se realiza y por lo tanto el *id* no se modifica.

De este modo el método *addProduct* siempre devuelve un *id* al método *addFav*, y este método realiza la siguiente acción, si el *id* es diferente de -1 mostrará un *Toast* con el siguiente mensaje, “Puede comprarlo más tarde accediendo a tus favoritos.” O por lo contrario mostrará un mensaje indicando “Esta oferta ya la tienes guardada entre tus favoritos.

Para un mejor entendimiento se realiza el siguiente esquema para tener una visión más general de lo explicado anteriormente.

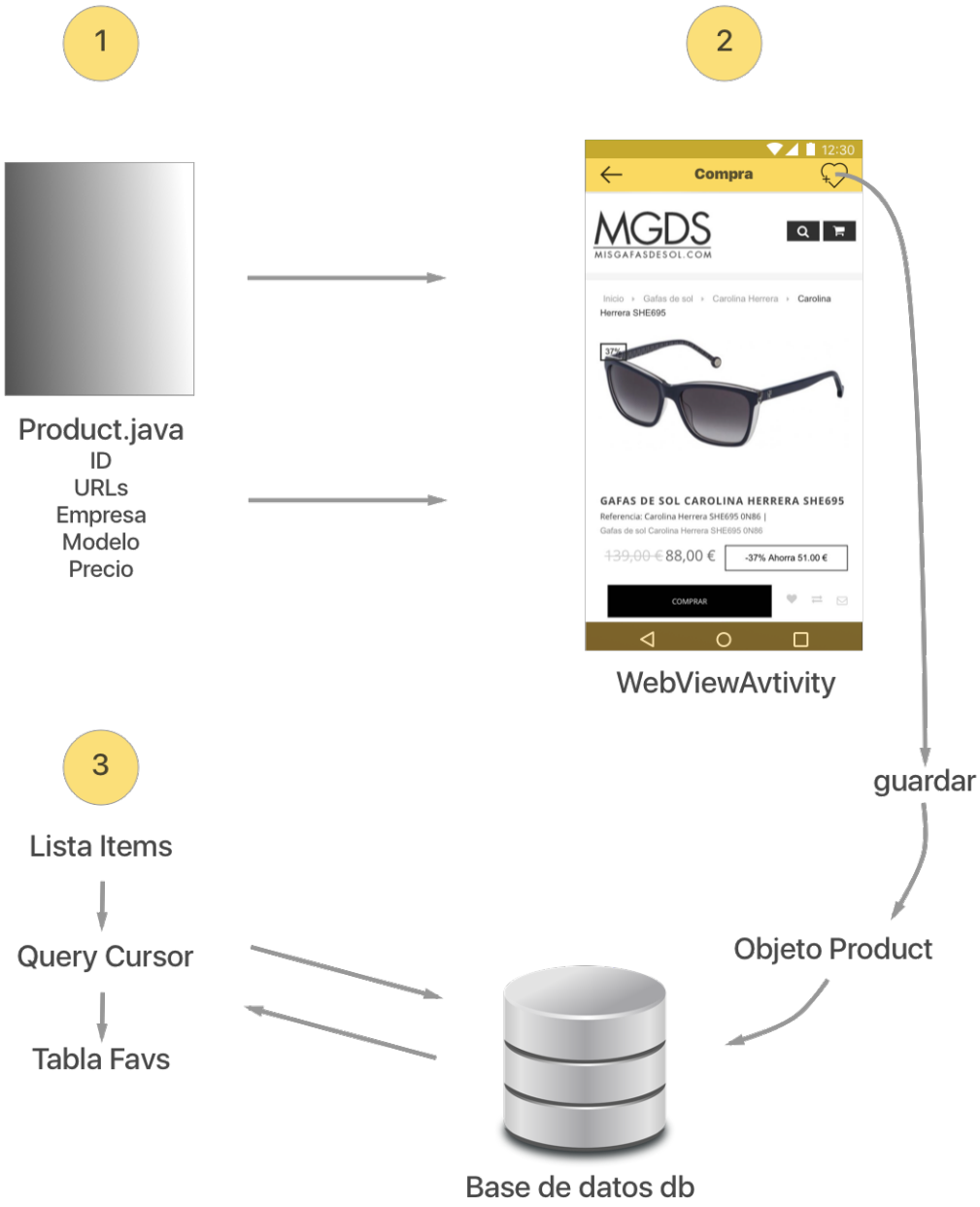

Figura 29. Esquema donde se muestra el flujo de los datos de la funcionalidad Añadir a favoritos.

También mencionar que para que los objetos *Product* puedan pasar de una actividad a otra la clase *Prouct.java* se implementa de la interfaz *Serializable*.
Una vez, se tiene la base de datos implementada, se busca la manera de poder visualizar correctamente la lista *Items* en la vista de *Mis Favoritos*.
Para ello se crea la clase *FavsAdapter.java* la cual hereda de un *ViewHolder*. Con esto se consigue utilizar un *RecyclerView* y la clase *FavsAdapter.java* se convierte en un adaptador para que el usuario pueda visualizar en la vista de *Mis Favoritos* mediante la clase *FavsActivity.java* una lista dinámica que se rellenará con la lista llamada *Items* mencionada anteriormente.
Si se observa el código adjuntado en el anexo, se puede observar que al constructor de la clase *FavsAdapter.java* hay que meterle como parámetros de entrada una lista de tipo *Product*, es decir un array rellenado de productos, y un *OnClickListener*. Este segundo parámetro tiene como función poder detectar en que parte de la pantalla el usuario tocará y como consecuencia poder asignar la acción que queremos como respuesta.

Esto es necesario implementarlo de este modo, ya que como se ha indicado es una lista dinámica y al variar dependiendo del número de productos que cada usuario tenga en su base de datos también variará la vista final que este percibirá.

Por otro lado, en la clase *FavsAdapter.java,* se debe instanciar la base de datos *db.* Posteriormente se obtiene la lista de productos con un método alojado en la clase *SQLiteFAvs.java* llamado *getAllProduct.*
Después de se debe asocia el adaptador con el XML correspondiente para que se visualice dónde y cómo se desee.
Por último, *en FavsActicity.java* encontramos un método llamado *OnCLick.* Este método se implementa para indicarle la acción que tiene que realizar el código cuando el usuario presión uno de los dos botones alojados en cada *Card* que muestra cada producto de la lista dinámica. Estos botones son *Comprar*, para redirigir al usuario de nuevo a la web oficial, o *Eliminar,* para eliminar de forma permanente la oferta que se detectó previamente.

Para eliminar un producto, lo primero que se hace es comprobar si hay registros en la *db*, si es así comprueba que el producto que es dependiendo del número de la fila donde se realizó el registro y con el método llamado *remove* de *List.java* indicamos que se elimine el producto de esa fila.

Para interactuar con el usuario se muestra un *Toast* indicando que la oferta ha sido eliminada.

Además, para no tener que cambiar de vista para comprobar que se ha eliminado correctamente, se utiliza un método llamado *notifyDataSetChanged* el cual actualiza de manera instantánea el adaptador y por consiguiente la vista actual que se está mostrando en pantalla y elimina al instante la *Card* del producto eliminado.

5.1.2 Requisitos no funcionales

En este apartado se explica el resto de funcionalidades interesantes, que pueden ser tanto para que se trabaje correctamente el resto de funcionalidades o para conseguir una mejor experiencia de usuario se decide añadir ciertos requisitos para hacer a la aplicación más atractiva e intuitiva.

- Utilización de datos alojados en un JSON.

Como ya se ha mencionado, la lista de productos que se pueden detectar, se encuentra guardad en un array dentro de un archivo llamado *produtcs.json*.

En primer lugar, es necesario, cargar dicho archivo en el código para su posterior uso. Para ello, se crea una clase llamada *Utils.java* en la carpeta *utils.*

En la cual se encuentran dos métodos:

- *LoadJSONFromAsset*, Método que se encarga de localizar dentro de la estructura del proyecto donde se encuentra el archivo *products.json* y posteriormente lo carga en java.
- *LoadProducts.* Método que extrae la información que esconde el archivo JSON y la convierte en formato java para poder utilizar dichos datos en variables que el código comprenda.

Para finalizar, se crea una clase llamada *Product.java,* donde se implementan métodos para la lectura y modificación de los datos ya convertidos a objetos de tipo *Product*. Como ya se ha mencionado, esta clase hereda de *Serializable* para poder utilizar los objetos de esta clase en cualquier otra.

- Notificaciones.

Como se explica en puntos anteriores el Modo Auto al activarse cambia la *StatusBar* y el icono del micrófono la vista *HomeActiviy.* Pero además se decide añadir un icono en dicha barra superior cuando este modo este activado. El icono será el logo de la aplicación y además se mostrará un mensaje indicando que se está activando el Modo Auto.

Para implementar esto se utiliza el método de java *notificationsBuilder.*
Cuando el código revisa que la preferencia *MODO_AUTO* está a *true,* llama a dicho método que indica que icono y mensajes se deben mostrar. También se le dota a la notificación de vibración con el método set*Vibrate* teniendo como parámetros de entrada los tiempos de vibración que se desean. Por último, se utiliza el método *notify* para lanzar la notificación tal y como se ha configurado. Este método pertenece a la clase *NotificationManager.java* que viene implementada por Android.

En las siguientes figuras se muestra la vista de *HomeActivity* con el Modo Auto ON y OFF.

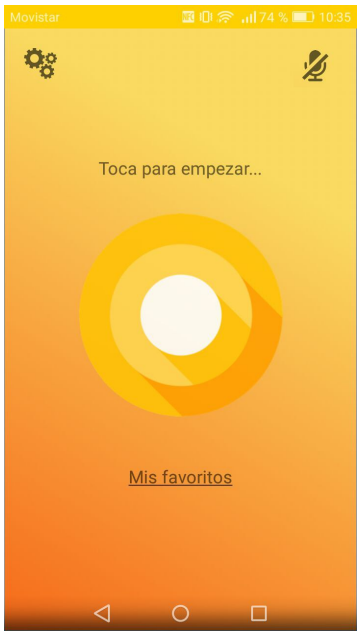

Figura 30. Pantalla Home con el Modo Auto OFF.

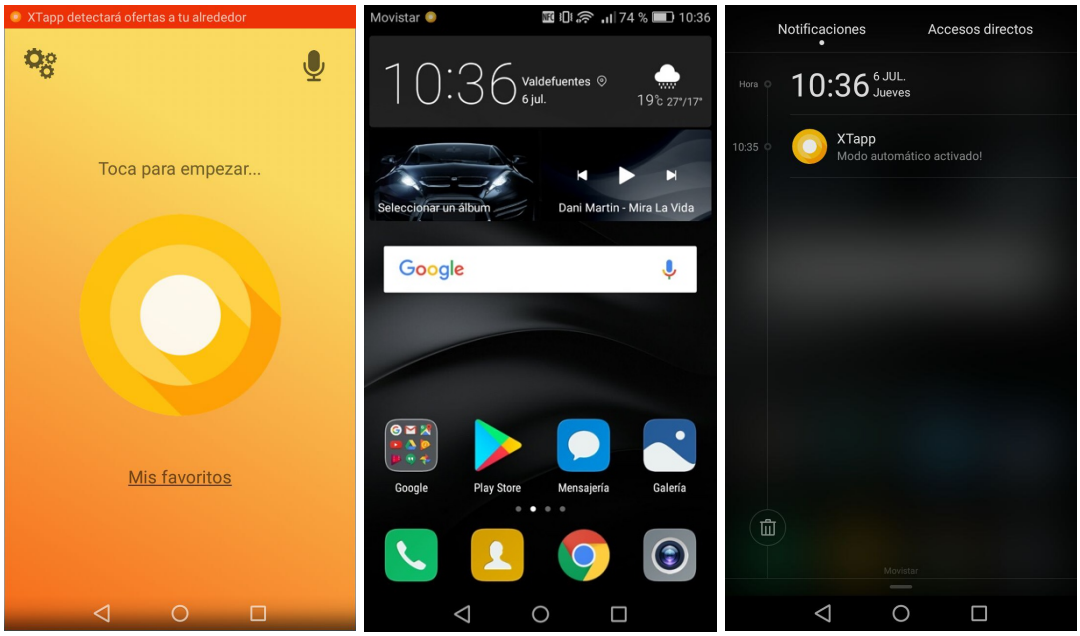

Figura 31. Vista de las pantallas cuando el Modo Auto está ON.

En el caso de que el usuario este fuera de la aplicación al presionar en el aviso del centro de notificación, pantalla tres de la figura 30, se navegará a la pantalla principal de **XTapp**.

Pensando en las detecciones que pueda hacer este modo, cuando el usuario lo tiene activado sin tener la aplicación en primer plano, se estudia la manera para que llegue un aviso al dispositivo.

Se llega a la conclusión que con la ayuda de la tecnología *FireBase*, se puede implementar en el proyecto para la recepción de este otro tipo de notificaciones.

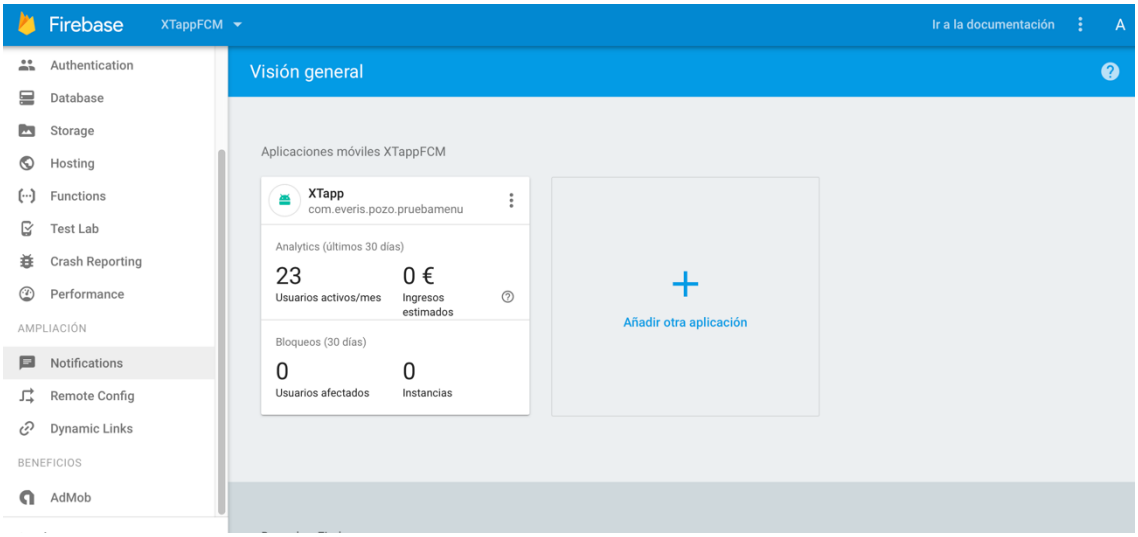

Figura 32. Interfazde Firebase.

FireBase es un servicio de google en el cual, por un lado, se debe registrar la aplicación en cuestión desde la interfaz del servicio, y por otro lado se debe introducir varias líneas de código en *Android Studio* para que funcione.

En el código de la aplicación que se encuentra en el anexo de este documento, se puede observar que en la carpeta *notificaciotions.*

Existen dos servicios:

- FCMService que extiende de FirebaseMessagingService.
- TokenService que hereda de FirebaseInstanceIdService.

Esto es necesario para que funcione el *Firebase Clouding Menssage* que será el encargado de hacer llegar este tipo de notificaciones al dispositivo del usuario cuando XTapp detecte un ultrasonido.

Además, en documento de *Gradle* del proyecto de *Android Studio* se deben añadir las siguientes dependencias:

- 'com.android..tools.build:gradle:2.3.3'
- 'com.google.gms:google-services:3.0.0'

A su vez, al principio del documento del módulo de la app de *Gradle* en *Android Studio* se debe añadir el puglin 'com.google.gms.google-services'
y en las dependencias del mismo documento se añade las siguientes:

- 'com.googles.firebase:firebase-core:10.0.1'
- 'com.google.firebase-messaging:10.0.1'

por último, en el archivo *Manifest* de *Android Studio* se debe indicar una vez más el uso de estos servicios.

Esto es necesario para que la aplicación entienda a que hacen referencia los servicios *FCMService* y *TokenService* y así poder compilarlo correctamente y posteriormente que funcione de manera correcta.

Visualmente el usuario percibirá las notificaciones del mismo modo que se muestran en la figura 30.

- Estilo de la *Card* del producto detectado.

La vista que se puede mostrar cuando se detecta una oferta, podía haber sido muy simple, con un cuadro de dialogo con dos botones hubiese bastado. Sin embargo, para hacer una aplicación más divertida y gráficamente más apetecible se decide implementar una visualización de la oferta utilizando una *Card*, elemento más atractivo que un simple cuadro de dialogo. Esta *Card* mostrará la imagen del producto que se detecta mediante el XTAudioBeacon.

Para elegir la omisión o la compra de dicha oferta, se diseñan dos botones en la parte inferior de la pantalla. Además se introduce un efecto a la *Card* que seguirá al movimiento que haga el usuario con el dedo al tocar la pantalla. El código que consigue esta funcionalidad se encuentra en la clase *Card.java* en la siguiente la carpeta *product*.

Con el método *onResolved* de esta clase, se le indica a la vista, los datos que tienen que visualizar y donde los puede encontrar.

Si se observa en detalle este método se encuentra el uso de *Glide.*

El resto de parámetros se indican de manera normal, pero las URLs precisan de la librería externa *Glide* para que se puedan visualizar correctamente.

Glide consigue pasar a formato imagen una URLs que en principio es formato texto. Para que esto funcione dicha URL debe tener alojada una imagen.

Como en el caso de *FireBase* la librería hay que indicarla en las dependencias del documento del módulo de *Gradle* de la app añadiendo 'com.github.bumtech.glide:glide:3.7.0'.

El resultado final se muestra en la siguiente figura.

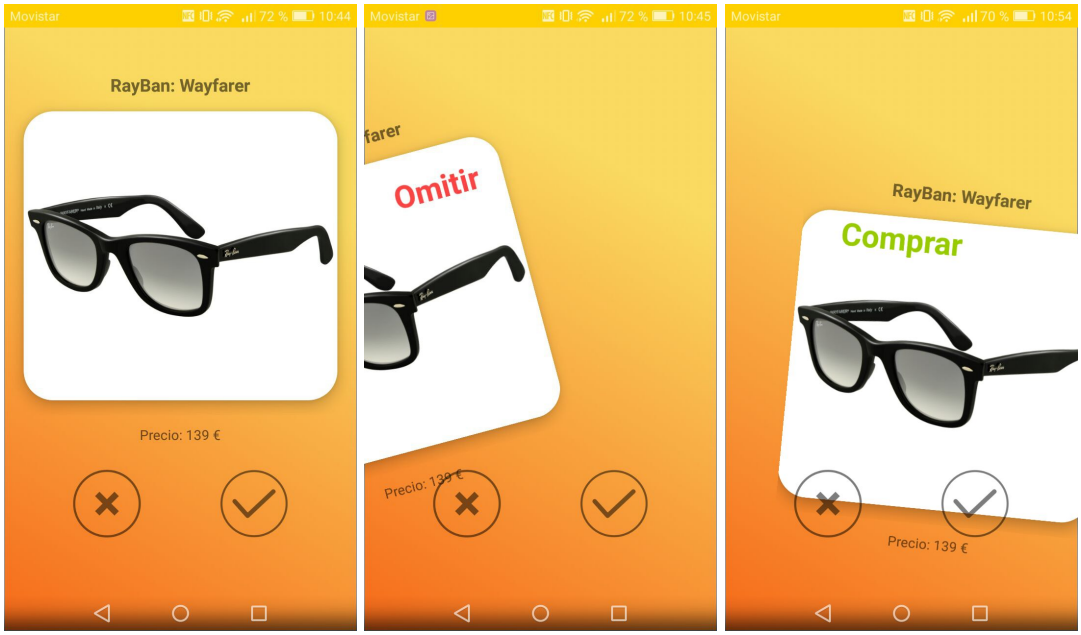

Figura 33. Vistas del producto detectado.

En la primera pantalla de la figura 32, se muestra la vista que aparece cuando se detecta el XTAudioBeacon. En la segunda pantalla se intenta transmitir lo que ocurre cuando el usuario desliza el dedo hacia la izquierda omitiendo el producto. Y en la tercera pantalla se intenta trasmitir lo que ocurre cuando el usuario desliza el dedo hacia la derecha accediendo a la compra del producto.

Como se ve en las tres imágenes, también se muestran los dos botones inferiores rechazo o aceptación comentados previamente.

5.2 Estudios Realizados

Este punto se centra en las pruebas que se han hecho de manera paralela al desarrollo para comprobar la efectividad de las funcionalidades que se estaban implementando.

5.2.1 Estudio acústico

Este estudio es imprescindible ya que el objetivo principal de esta aplicación es que detecte los XTAudioBeacons.

Como primer punto, el estudio ha consistido en generar con Matlab tonos puros a frecuencia concretas. Estas frecuencias van desde los 16kHz a los 22kHz.

Esta tarea se realiza del siguiente modo:

Comandos Matlab para generar tonos:

Se procede a generar diferentes tonos para posteriormente utilizarlos en la fase experimental.

Se crea un fichero de *Matlab* llamado *tono.m* con la siguiente función:

```
function s=tono(A,f,fs,dur)
%          s=tono(A,f,fs,dur)
%          genera un tono de:
%                    A      : Amplitud en voltios
%                    f(Hz)  : Frecuencia
%                    fs (Hz): Frecuencia de muestreo.
%       dur    : Duracion (seg)
nmax=dur*fs;
n=0:ceil(nmax);
s=A*cos(2*pi*n*f/fs);
%--------------------------------------------------------------------------
>> A=0.5;            //Amplitud del tono

>> fs=44100;         //Frecuencia de muestreo

//Frecuencia de cada tono
>> f16=16000;
>> f17=17000;
>> f18=18000;
>> f19=19000;
>> f20=20000;
>> f21=21000;
>> f22=22000;
>> f23=23000;
>> f24=24000;
>> dur=1;            //Duración del tono

//Llamada a la función tono, para generarlos.
>> s16=tono(A,f16,fs,dur);
>> s17=tono(A,f17,fs,dur);
>> s18=tono(A,f18,fs,dur);
>> s19=tono(A,f19,fs,dur);
>> s20=tono(A,f20,fs,dur);
>> s21=tono(A,f21,fs,dur);
>> s22=tono(A,f22,fs,dur);
>> s23=tono(A,f23,fs,dur);
>> s24=tono(A,f24,fs,dur);

//Escucha cada tono
>> sound(s16,fs)
>> sound(s17,fs)
>> sound(s18,fs)
>> sound(s19,fs)
>> sound(s20,fs)
>> sound(s21,fs)
>> sound(s22,fs)
```

```
>> sound(s23,fs)
>> sound(s24,fs)
>> sound(s24,fs)
>> sound(s23,fs)
>> sound(s24,fs)
```

Llegados a este punto, se observa que a partir de 24.000 kHz se vuelve a percibir el sonido, esto no es posible, con lo cual se llega a la conclusión que es debido a la frecuencia de muestreo elegida, que, al ser 44.100 kHz, a más de 22.000 kHz los tonos sufren *aliasing*.

Por lo tanto, se generan los nuevos tonos a una frecuencia de muestreo de 48.000 kHz.

Ahora, para visualizar su espectro se crea otro fichero de *Matlab* llamado *espectro.m* donde se implementamos la siguiente función:

```
function y=espectro(x,fs);
%   Esta funcion estima la densidad espectral de potencia.
%   mediante el periodograma,
%
%
%   function y=espectro(x,fs)
%
%   x      :Vector sobre el que deseo estimar el espectro.
%   fs     :Frecuencia de muestreo
%
x=x(:);
ejef=fs/2;

if length(x)<1024 res=1024; else res=length(x);end
y=(abs(fftshift(fft(x,res)))/res).^2;
semilogy(linspace(-ejef,ejef,length(y)),y);grid on;zoom on;
set(gca,'XLim',[-ejef ejef]);
axis([-ejef ejef 0 max(y(:))]);
title('Espectro');xlabel('Frecuencia');ylabel('Magnitud');
setptr(gcf,'glass');warning off;

%------------------------------------------------------------------------
```

```
//Se visualizan todos los espectros:
>> espectro(s16,fs);
>> espectro(s17,fs);
>> espectro(s18,fs);
>> espectro(s19,fs);
>> espectro(s20,fs);
>> espectro(s21,fs);
>> espectro(s22,fs);
>> espectro(s23,fs);
>> espectro(s24,fs);
```

Se comprueba con cada espectro que todos los tonos tienen la frecuencia correcta, como ejemplo se muestra la siguiente figura con el espectro *s20*.

Si se hace zoom se observa claramente que es un tono a 20.000 kHz.

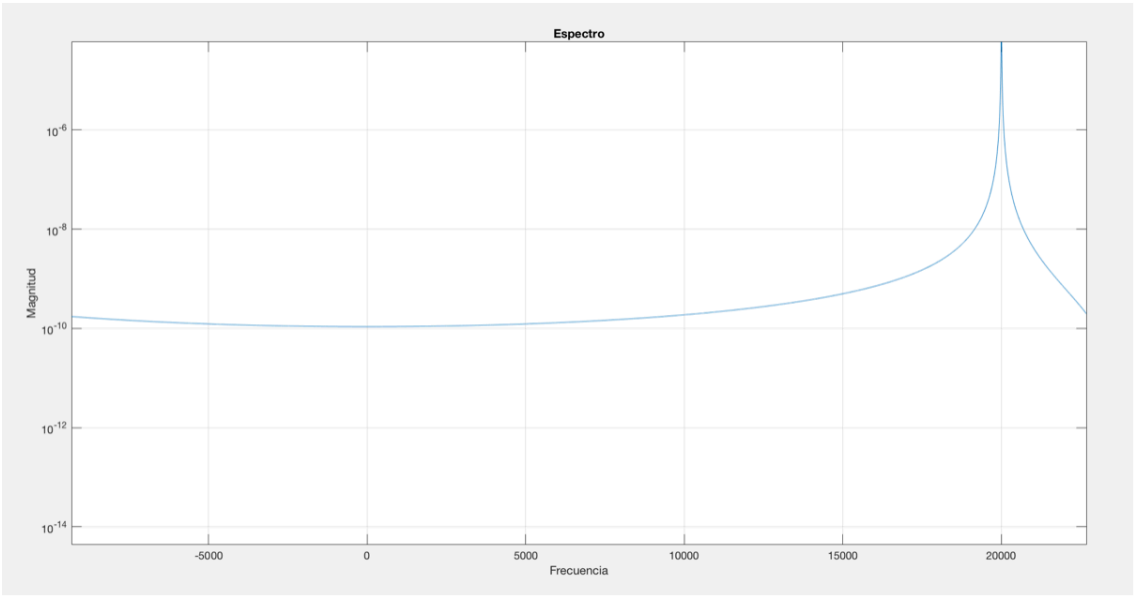

Figura 34. Zoom espectro a 20.000kHz.

Posteriormente se guarda cada uno de los tonos generados. Se elige un formato de audio tipo .wav, ya que admite tonos a frecuencias más altas.

```
>> audiowrite('s16.wav',s16,fs);
>> audiowrite('s17.wav',s17,fs);
>> audiowrite('s18.wav',s18,fs);
>> audiowrite('s19.wav',s19,fs);
>> audiowrite('s20.wav',s20,fs);
>> audiowrite('s21.wav',s21,fs);
>> audiowrite('s22.wav',s22,fs);
>> audiowrite('s23.wav',s23,fs);
>> audiowrite(‘s24.wav’,s24,fs);
```

Para comprobar si tanto la recepción como la emisión entre dispositivos se puede efectuar y se registran las frecuencias tal y como se han generado se utilizan varias aplicaciones de *Play Store:*

- Spectrum Analizer:

Figura 35. Información de *Play Store* sobre *Spectrum Analizer.*

- Spectrum Analyze:

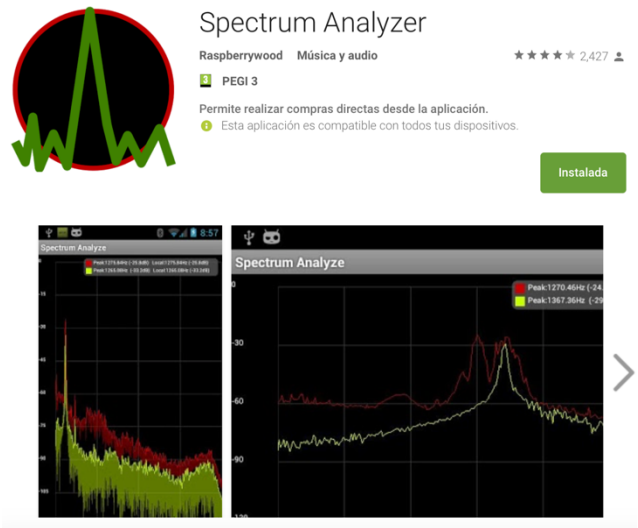

Figura 36. Información de *Play Store* sobre *Spectrum Analize.*

- ProSpec Lite:

Figura 37. Información de *Play Store* sobre *ProSpec Lite.*

Tras comprobar el funcionamiento de las tres alternativas se decide utilizar la segunda opción. Figura 36.
Se elige *Spectrum Analyze* por sus mejores resultados en la obtención de los espectros frente al resto de aplicaciones.

En las siguientes figuras se muestran los resultados de dichos espectros registrados en un Smartphone, siendo el emisor un ordenador portátil con los altavoces que lleva integrados de fábrica.

Tonos utilizados:

En las figuras que se muestran a continuación la información más importante se encuentra en el cuadro situado arriba a la derecha de cada imagen y en el mismo espectro para comprobar que está recibiendo exactamente el dispositivo receptor.

- Frecuencia a 16 kHz

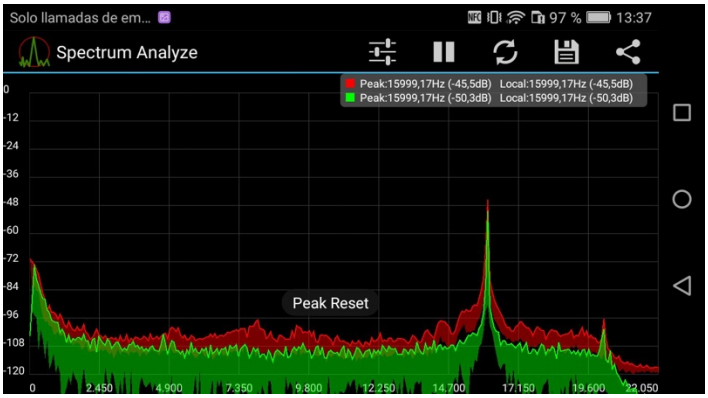

Figura 38. Espectro frecuencial del tono s16.

- Frecuencia a 17 kHz

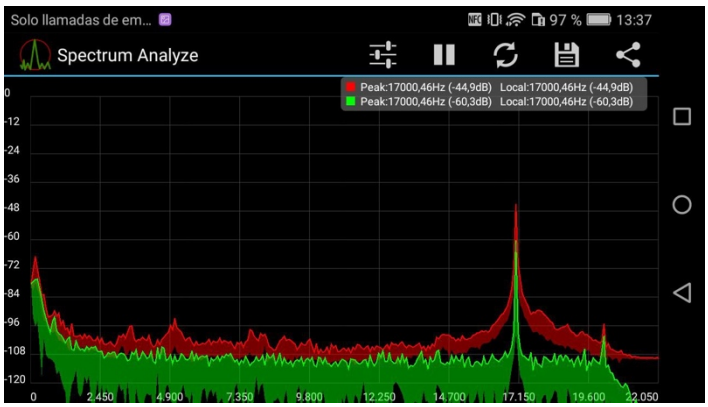

Figura 39. Espectro frecuencial del tono s17.

- Frecuencia a 18 kHz

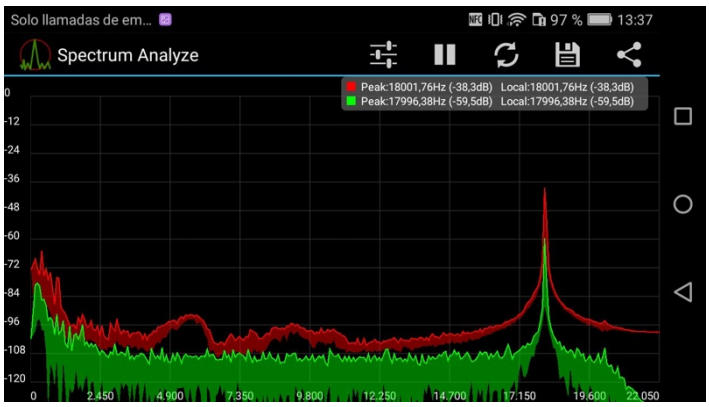

Figura 40. Espectro de frecuencias del tono s18.

- Frecuencia a 19 kHz

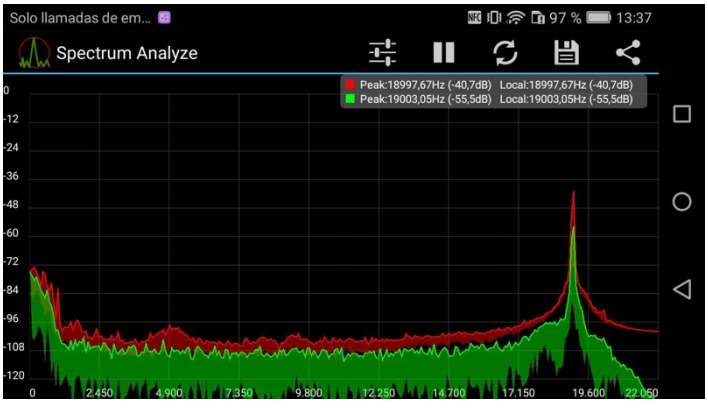

Figura 41. Espectro de frecuencias del tono s19.

- Frecuencia a 20 kHz

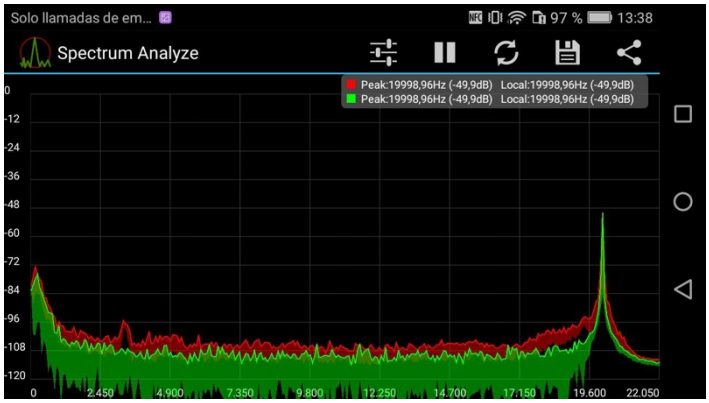

Figura 42. Espectro de frecuencias del tono s20.

- Frecuencia a 21 kHz

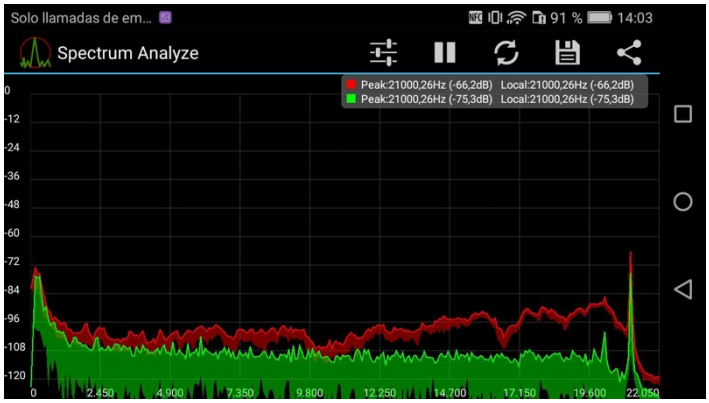

Figura 43. Espectro frecuencial del tono s21.

- Frecuencia a 22 kHz

Figura 44. Espectro de frecuencias del tono s22.

Se observa en las figuras anteriores que la recepción es la esperado menos en el casi del tono *s22.*
Como a 22 kHz no se registra la captación del tono, se generan con la función *tono(A,fx,fs,dur);* siendo *fx* frecuencias con intervalos de 100 Hz para determinar con más exactitud cuál es la frecuencia limite que se consigue recibir.

Tras realizar varios intentos se llega a la conclusión que el móvil no es capaz de registras frecuencias superiores a 21.500 kHz. Este dato es de vital importancia, ya que ahora se puede afirmar que no es eficiente generar en un futuro tonos a frecuencias por encima de este valor porque la aplicación no conseguirá realizar la detección correctamente.
Como segundo punto, se comprueba la detección de los XTAudioBeacons y comprobar si diferencia uno de otro y se consigue mostrar el producto asociado a cada uno.
En este punto, se emiten los audios sin ningún ruido alrededor, es decir, sin ruido ambiente, ni canciones, etc.
Al intentar realizar la acción de detectar, se comprueba que funciona correctamente.
El siguiente paso es comprobar la distancia, para esto además se implementa un método llamado *IndicateProximity* en la clase *Listenning.java*. Con este método se consigue cambiar de color una animación implementada anteriormente que aparece en esta vista para indicar al usuario a que distancia esta del emisor.
Esto se consigue calculando la amplitud de la reproducción. De este modo funciona sin problemas, aunque tiene como hándicap que puede ser modificada subiendo y bajando el volumen del XTAudioBeacon, es decir no hace un cálculo real de la distancia al emisor.
Independiente de este método adicional, se comprueban con la distancia máxima a la que se puede recibir la reproducción es aproximadamente de 10 metros. Esta distancia es posible que varíe dependiendo del entorno acústico que exista en el momento de la detección.
Como tercer punto, lo que se intenta es insertar el XTAudioBeacon en un anuncio de televisión.
Para esto, se descargan dos anuncios de YouTube en formato .wav y con *iMovie* se crean dos videos con diferentes XTAudioBeacons insertados.
Un ejemplo de la creación de estos videos dotados con los ultrasonidos es el mostrado a continuación.

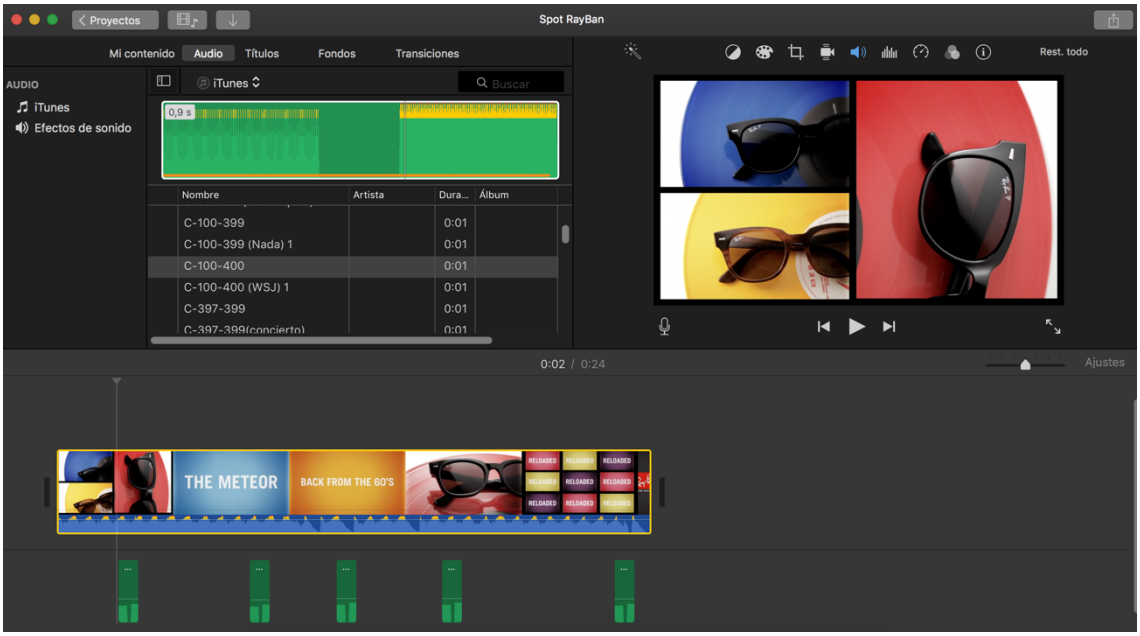

Figura 45. Edición de los anuncios insertando el XTAudioBeacon con iMovie.

En esta figura, es la primera vez donde se muestra la forma de onda que tiene un XTAudioBeacon en la parte superior izquierda de la imagen de color verde.

En la parte inferior se aprecia por un lado el video con su sonido en color azul, siendo este una canción que llevaba el anuncio, y por otro lado en color verde, los XTAudioBeacons añadidos en varios puntos del video.

Se ha intentado situar dichos audios en las zonas donde el sonido que llevaba el video tenía una amplitud más baja para que se interrumpieran lo menos posible y que la detección fuera más óptima.

Además, se subió la ganancia del sonido del anuncio y situándolo en un 112%, siendo 100% la ganancia que ya llevaba, y se bajó las de los XTAudioBeacons hasta situarlo en 14%.

Esto se decide hacer para que, cuando se reproduzcan los ultrasonidos en altavoces convencionales, no sufran *clipping,* ya que esto es un problema porque molestaría mucho a los usuarios por el ruido que puede producir el altavoz, además de poder estropear el emisor.

Tras intentar la detección de este modo, se comprueba que se recibe correctamente, en la mayoría de puntos del anuncio donde se encuentran insertados los XTAudioBeacons.

5.2.2 Estudio con materiales

Este apartado se centra en el estudio de la recepción y por consecuencia de la detección de los XTAudioBeacons cuando el dispositivo del usuario se encuentra en el interior de los bolsillos de los pantalones, bolsos, o cualquier prenda o accesorio que utiliza la sociedad de manera general para guardar el Smartphone.

Como el modo auto se encuentra en desarrollo en el momento de realizas este estudio, se utiliza la funcionalidad principal que al final y al cabo realiza la detección de la misma manera.

La primera prueba consiste en introducir el dispositivo en el bolsillo del pantalón. Al iniciar la detección a 1 metro del emisor se recibe la oferta sin problema, pero en el momento de incrementar la distancia el XTAudioBeacon y no é reconocido por la aplicación.

La segunda prueba se realiza introduciendo el terminal móvil en un bolso. La consecuencia es que la distancia se reduce a la mitad que en el caso anterior para que la aplicación consiga realizar la detección.

Tras investigar el caso, se llega a la conclusión de que esto tipo de frecuencias sufren mucha absorción por parte de materiales textiles, como algodón, tejido vaquero, y los convencionales que se utilizan para la fabricación de ropa.

5.3 Herramientas y dispositivos utilizados

En este apartado se enumeran las tecnologías, software y hardware utilizados para el desarrollo del TFG.

- Android Studio: Entorno de desarrollo.
- Genymotion: Simulador de dispositivos Android.

- https://bubbl.us: Página web para la creación de mapas mentales.
- Sketch: Sofware de diseño para la creación de las vistas de la aplicación.
- Sketch ToolBox 2: Plugin del software anterior para integrarlo en Marvel.
- MarvelApp: Página web para la creación del prototipo.
- Matlab: Sofware para la creación de tonos puros.
- XTUltrasonicsDemo: Aplicación de Android similar para conseguir la detección de ultrasonidos.
- Spectrum Analize: Aplicación de Android para comprobar los espectros de frencuencia.
- Pages: Sofware para la redacción de la parte de los estudios.
- Microsoft Word: Sofware para la redacción de la memoria del TFG.
- SourceTree: Sofware para utilizar Git como control de versiones.
- Numbers: Sofware para la creación del diagrama temporal.
- Firebase: Interfaz para implementar ciertas funcionalidades de la app.
- Fabric.io: Interfaz para implementar ciertas funcionalidades de la app.
- Huawei P8 Lite: Smartphone para comprobar el funcionamiento en general.
- MacBook Pro: Ordenador portátil para desarrollar todo el TFG.
- iPhone 7: Smartphone utilizado como emisor.

5.4 Resultados

5.4.1 Resultados esperados

Se espera que la aplicación consiga detectar los XTAudioBeacons insertados en varios anuncios en lugares con un ruido ambiente moderado, realizando las emisiones desde varios dispositivos.

5.4.2 Resultados obtenidos

La aplicación detecta el evento sonoro como se esperaba. En condiciones normales (ofician de trabajo) con el Huawei P8 Lite, se registran los ultrasonidos a una distancia máxima de diez metros, teniendo el volumen de MacBook al máximo sin que distorsione la señal de salida.

La aplicación se ha probado en bares llenos de gente y parkings de centros comerciales a hora punta para comprobar si funcionaba, la distancia máxima de recepción variaba por el ruido de alrededor, pero en todas las condiciones la aplicación conseguía realizar la detección esperada.

Bien es cierto que la mayoría de situaciones de este tipo el Smartphone debía estar a dos metros como máximo para que se lograse el objetivo.

Con ruido excesivamente elevados **XTapp** no conseguía la detección.

Capítulo 6. Consecuencias para la salud

Apartir de los 25 años se empieza a perder audición a altas frecuencias. Por lo tanto, los XTAudioBeacons, al no ser perceptibles por el oído humano, no son perjudiciales. Además, la potencia acústica trasmitida es muy pequeña y por tanto se puede afirmar que esta tecnología es inofensiva.

En cambio, los animales si pueden verse afectados ya que su cerebro sí que es capaz de procesar esa información. Sería necesario un estudio médico más exhaustivo para determinar cuál sería el nivel de peligro para ellos y las consecuencias que podrían tener.

Capítulo 7. Conclusiones y propuesta de trabajo futuro

Este proyecto ha sido una enseñanza continua y con él he podido conocer como una empresa gestiona proyectos reales de clientes importante y los pasos que se siguen de manera eficaz desde que el cliente comenta el producto que quiere hasta que se le hace entrega, pasando por todos y cada uno de los procedimientos necesarios para que todo salga según lo esperado.

Como trabajo futuro, voy a seguir mejorando y haciendo más robusta la aplicación junto a profesionales de la empresa donde he podido realizar el TFG, se solucionarán problemas de *crasheos* inesperados, se dará funcionalidad al menú ya diseñado, se estudiará el modo de sacar partido al modo auto que está en desarrollo en estos momentos y se intentará vender la idea a grandes empresas para dar un paso más allá y apostar porque la gente llegue a utilizarla.

Para solucionar muchos de los errores existentes hoy en día en la aplicación se va a implementar *Crashlytics* que monitoriza la aplicación y te notifica los errores que han ocurrido mientras los usuarios la han utilizada, y además se va a subir a *Fabric.io* para que cualquier usuario de *Android* y compañeros de Everis puedan utilizarla, siempre de manera controlada, para tener mucha más información de los errores existentes y seguir mejorando.

Por otro lado, la reacción que realiza **XTapp** como se ha comentado, está enfocada hacia el marketing digital, pero se podría utilizar para realizar pago, mostrar ayuda de algún problema en concreto o cualquier idea que pueda surgir en tiempos futuros, así que estoy convencido que haciendo bien las cosas y cuidando los detalles se podría sacar mucho partido a este tipo de comunicaciones.

Capítulo 8. Bibliografía

Documentación utilizada para la realización del TFG.

Curso android:
http://www.sgoliver.net/blog/interfaz-de-usuario-en-android-controles-basicos-i/

Creación prototipos:
https://marvelapp.com/dashboard/

Tutorial creación APP:
http://www.jtech.ua.es/cursos/apuntes/moviles/daa2013/wholesite.pdf
https://codigofacilito.com/videos/programacion_android_event_onclicklistener

Permisos Android:
Como meter permisos en Android Studio:
https://androidstudiofaqs.com/tutoriales/dar-permisos-a-aplicaciones-en-android-studio
Lista de permisos de una APP:
https://developer.android.com/reference/android/Manifest.permission.html?hl=es
Permiso Bateria Estadísticas:
https://developer.android.com/reference/android/Manifest.permission.html?hl=es#BATTERY_STATS
Permiso para capturar la salida del audio:
https://developer.android.com/reference/android/Manifest.permission.html?hl=es#CAPTURE_AUDIO_OUTPUT
Grabar audio:
https://developer.android.com/reference/android/Manifest.permission.html?hl=es#RECORD_AUDIO
Ejecución en segundo plano:
https://developer.android.com/reference/android/Manifest.permission.html?hl=es#RUN_IN_BACKGROUND

Funcionamiento de espionaje con uXDT:
http://m.forocoches.com/foro/showthread.php?t=5601084
https://www.xataka.com/privacidad/234-aplicaciones-de-android-estan-rastreando-la-actividad-de-sus-usuarios-a-traves-de-senales-ultrasonicas
https://andro4all.com/2017/05/aplicaciones-android-espiar-usuarios-ultrasonidos

Llamar a mi APP con OK google:
https://developers.google.com/voice-actions/system/?hl=es-419

Utilizar servicios Google:
https://www.ladrupalera.com/drupal/desarrollo/javascript/como-usar-una-api-de-google-con-autenticacion-traves-de-oauth2

Compartir mi APP:
https://firebase.google.com/docs/invites/android?hl=es-419

Captar audio y grabarlo:
https://developer.android.com/guide/topics/media/mediarecorder.html

Simil con Beacons/Balizas bluetooth:
https://thevalley.es/blog/que-son-los-beacons-y-cual-es-su-potencial

Técnicas de publicidad personalizada:
http://lapastillaroja.net

APP similar:
(codigo, app de ejemplo, iBeacons de ejemplo)
https://github.com/jamrader/XTAudioBeacons

Video ejemplos:
https://vimeo.com/200193202

Espectáculo de luces:
https://www.youtube.com/watch?v=UkxqUhp2RCk

Diseño app:
http://www.hermosaprogramacion.com/2016/02/textinputlayout-en-android-material-design/

Para perfeccionar el diseño al máximo:
http://appdesignbook.com/es/contenidos/preparando-los-archivos/

Uso de RecyclerView para las Cards:
https://developer.android.com/training/material/lists-cards.html?hl=es
https://code.tutsplus.com/es/tutorials/getting-started-with-recyclerview-and-cardview-on-android--cms-23465
http://www.sgoliver.net/blog/interfaz-de-usuario-en-android-cardview/

Git:
https://git-scm.com/book/es/v1/Fundamentos-de-Git-Guardando-cambios-en-el-repositorio

Funcionando en segundo plano:
http://www.sgoliver.net/blog/tareas-en-segundo-plano-en-android-i-thread-y-asynctask/

Capítulo 9. Anexos

En este anexo se detalla de manera escrita una lista con los archivos que se pueden encontrar adjuntados a este proyecto en la carpeta *Ficheros Anexos* y los proyectos a los cuales se tiene acceso vía URL.

- Proyecto de conceptualización y prototipo.

  https://marvelapp.com/37jb8e9

- Proyecto Android Studio con código fuente.
- Audios generados con Matlab.
- XTAudioBeacons utilizados para la detección.
- Bitbucket para la versión de pruebas con Git.

  https://apozop@bitbucket.org/apozop/xtapp.git

- Mapa mental final.

  https://bubbl.us/NDAyODE1OS84MTU0NDc2LzczZmQ4MGRmMThmMjRjOWI0ZmYyOWU0ZDc0MjUzMjIw-X?utm_source=shared-link&utm_medium=link&s=8154476