ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Máster

Aplicación Android de Acceso a Servidor de Datos Multimedia

AUTOR: Enrique Ant. Rivas Rodríguez

DIRECTOR: Fernando Cerdán Cartagena

CODIRECTORES: Juan Suardiz Muro, Diego García Sánchez

Julio 2012

Autor	Enrique Ant. Rivas Rodríguez
E-mail del Autor	enr.rivas@gmail.com
Director	Fernando Cerdán Cartagena
E-mail del Director	fernando.cerdan@upct.es
Título de TFM	Desarrollo de una Aplicación Android para la gestión y venta de contenido multimedia

Resumen

El proyecto se basará en el desarrollo de una aplicación para el sistema operativo Android bajo el nombre oliva, la cual consiste en la gestión y venta de contenido multimedia alojados en un servidor como son, Imágenes, Documentos, Audio y Video para kioscos ubicados en lugares específicos como podrían ser lugares de interés turísticos.

A modo de ejemplo esta aplicación podría ser fácilmente utilizada en museos donde al momentos de visitarlos este le ofrezca a los visitantes la ventas de ciertos contenidos multimedia como bien podría ser unas de sus presentaciones del momento (video), la impresión o descarga de alguno de los cuadros famosos o que le gusten al visitante (imágenes), en vez de utilizar folletos impresos, ya estos estén presente en la aplicación y se reduce el uso de papel (documentos) o incluso la descarga de algún tipo de audio que represente algo atractivo para el visitante del museo.

Titulación	Máster en Tecnología de la Información y Comunicaciones
Departamento	Tecnología de la Información y Comunicaciones
Fecha de Presentación	Julio2011

Agradecimientos

A mis padres, Enrique Rivas y Juliana Rodríguez que no importando la situación y mi forma de ver las cosas estuvieron ahí para apoyarme. A todos mis hermanos, en especial a Gioribel y Luis David Rivas los cuales sin darse cuenta me han forzado hacer una mejor persona. A la familia Pagán Rivas por el trato incondicional que me han brindado. A mi familia en general, especialmente mis tíos por brindarme su apoyo y creer en mí. A mi tutor de proyecto Fernando Cerdán, por el apoyo y las facilidades brindadas. A Diego García, Carlos Bermúdez, Miguel Ángel García por todas las ideas aportadas y sus formas peculiares de ver las cosas. A esos amigos que se mantuvieron presente a pesar de la distancia. A mis nuevos amigos por hacer de mi estancia lejos de casa resultase tan agradable.

Índice

Capítulo 1.

Introducción

Las nuevas tendencias creadas debido al uso masivo de teléfonos inteligentes y Tablets, han brindando toda una nueva forma de cómo queremos consumir el contenido multimedia actual, es mucho más factible consumir contenido virtual en vez de contenido físico. Teniendo como gran ventaja el contenido virtual sobre el físico la movilidad y portabilidad. Aprovechando esta tendencia existen hoy en el mercado muchas aplicaciones para la gestión y ventas de contenido multimedia, las cuales han estado dando buenos resultados y es de estas aplicaciones que surge la idea de crear éste proyecto de nombre Oliva, el cual es una aplicación desarrollada para el sistema operativo de teléfonos móviles Android. Oliva consiste en la gestión de contenidos multimedia alojados en un servidor remoto para su visualización y / o la venta de los mismos.

1.1 Motivación

El mercado de teléfonos móviles inteligentes está cada vez más en aumento, reinando dos sistemas operativos en estos, IOS y Android. Por un lado IOS actualmente tiene un 20% del mercado mientras que Android más de un 75% del mercado y el restante lo conforman otros sistemas operativos. Esta gran presencia del sistema operativo Android, es lo que motivó el deseo de investigar cómo desarrollar una aplicación para Android.

Después del interés de desarrollar una aplicación, es necesario encontrar o determinar que aplicación desarrollar, la cual debe ser atractiva para que el intereses no se pierda mientras se esté realizando las investigaciones pertinentes, debido a esto la propuesta de crear una aplicación para la gestión y ventas de contenidos multimedia alojados en un servidor remoto fue de lo más

atractiva ya que se estaría desarrollando una aplicación que se podía dividir en tres grandes fases. Una interfaz atractiva para el usuario. Gestión de Pago. Manejo de Servidor de Datos.

1.2 Objetivos del Proyecto

Los objetivos del proyecto están muy ligado a la motivación de mismo ya que lo se pretende a grande rasgo es aprender a desarrollar aplicaciones para el sistema operativo de móviles Android donde se pudiera interactuar con un servidor o servidores externo, es decir crear aplicaciones del tipo cliente servidor y en la cual se pudiese implementar uno o varios sistemas de pago.

Pero tan solo aprender a desarrollar la aplicación no completa los objetivos de este proyecto sino desarrollar una aplicación funcional que interactué con el usuario, se conecte a un servidor, descargue su contenido y el usuario pueda visualizarlos y pagar por ellos.

1.3 Estructura de la memoria

La estructura del proyecto es la siguiente: En el capítulo 2 abarcará el tema relacionado con Android, ¿Qué es?, Un breve historia y como se estructuran las aplicaciones. Los capítulos 3 y 4 explican el diseño de la aplicación, sus funcionalidades y la implementación de la aplicación. El capitulo 5 nos muestra un ejemplo de unas de las funcionalidades del prototipo creado de la aplicación y el capitulo 6 contiene las conclusiones obtenidas al realizar el proyecto y las posibles mejoras a realizar.

Capítulo 2.

Android

"Android es un sistema operativo móvil basado en Linux, que junto con aplicaciones middleware está enfocado para ser utilizado en dispositivos móviles como teléfonos inteligentes, tabletas, Google TV y otros dispositivos. Es desarrollado por la Open Handset Alliance, la cual es liderada por Google. Este sistema por lo general maneja aplicaciones como Market (Mercado) o su actualización, Google Play" [4].

2.1 Estructura de Android

Android se puede ver como un conjunto de de software que incluye un sistema operativo, software intermedio que trabaja al servicio de las aplicaciones que se encuentran en el nivel más alto y algunas aplicaciones que ya vienen incluidas desde un inicio con el sistema operativo. A modo de ejemplo una de estas aplicaciones por defecto es Android Market o Google Play en su actualización, la tienda en donde podemos comprar o descargar gratuitamente aplicaciones que son ofrecidas por terceros.

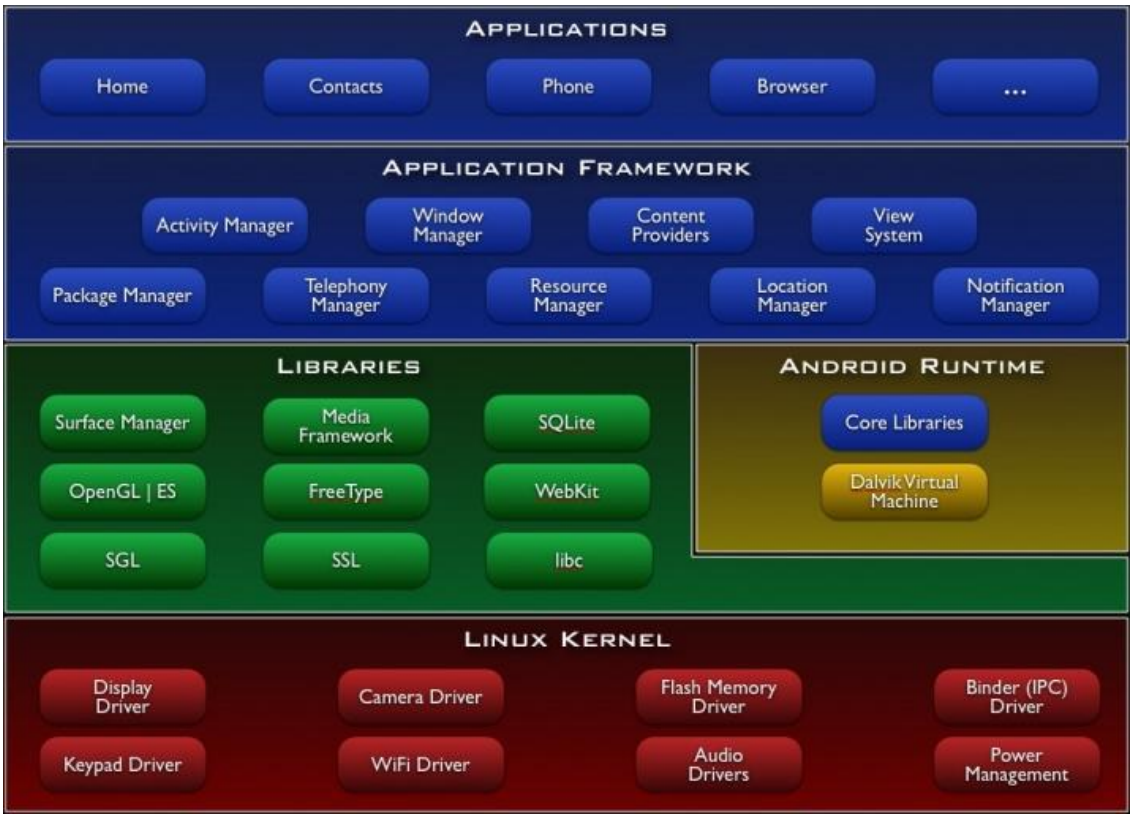

Figura 2.1 Estructura Interna Android

En la figura 1.1 podemos apreciar la estructura interna de Android [5], donde podemos ver que el núcleo del sistema operativo es una modificación del núcleo de Linux. En este núcleo se encuentran los drivers que permiten la comunicación entre el hardware específico de cada dispositivo que implementa Android y el sistema operativo. En el siguiente nivel se encuentran todas las librerías accesibles a la hora de querer programar aplicaciones, las cuales para su buen funcionamiento hacen uso de los drivers implementados. Unas de estas librerías pueden ser OpenGL ES, SQLite, SSL, WebKit, FreeType etc.

Dentro de Android encontramos la conocida maquina Dalvik, ya que las aplicaciones que se ejecutan sobre Android se ejecutan en una instancia de la máquina Dalvik. Cada instancia es independiente lo que conlleva la aplicación se ejecuta de forma cerrada. Este es un buen mecanismo de seguridad debido a que no es posible entrometerse con la ejecución de una aplicación. De igual forma, los recursos asignados a cada aplicación se encuentran en un fragmento de memoria privada solamente accesible desde dentro de la aplicación.

En la siguiente capa, *Framework Application*, se encuentran unas series de componentes los cuales son utilizados por las aplicaciones para realizar ciertas funciones determinadas. Entre estos componentes podemos encontrar, por ejemplo, *Notification Manager*, el cual recibe notificaciones de las aplicaciones y las muestra al usuario a través de la barra de notificaciones.

Otro ejemplo podría ser el *Activity Manager*, que nos permite gestionar las actividades de cada aplicación.

Debido a que los componentes que conforman el *Framework Application* manejan información sensible para el usuario, información sensible son datos personales los cuales son muy comunes en los dispositivos móviles, Android cuenta con un mecanismo de permisos con el propósito de mantener la información de forma segura. A Raiz de este mecanismo de seguridad, al momento de querer instalar una aplicación está le autorización al usuario para poder acceder a los componentes que serán necesarios del *Framework Application* para el buen funcionamiento de la aplicación. En caso de que el usuario no acepte estos permisos la aplicación no será capaz de utilizar los componentes que necesita.

El siguiente nivel que podemos apreciar en la figura 1.1 es *Applications* o aplicaciones. En este nivel como su nombre lo dice se encuentran las aplicaciones, estas pueden ser creadas tanto por desarrolladores externos como por Google. De las aplicaciones creadas por Google frecuénteme ya vienen pre instaladas en el sistema operativo.

2.2 Historia de Android

Android es un sistema operativo desarrollado por Google posteriormente a la compra de la empresa Android Inc., de donde obtuvo capital humano para la creación del sistema operativo. Debido a que es Google quien desarrolla en estos momentos a Android este debe poseer una total integración con los servicios que ofrece Google.

Figura 2.2 Logotipo Android

El objetivo principal de Google a la hora de crear y mantener Android es crear un ecosistema móvil estándar y abierto, con el propósito de satisfacer las necesidades de fabricantes de dispositivos móviles y tablets.

Desde su salida al mercado en el año 2008, Android ha ido escalando puestos en el ranking de ventas, hasta llegar a ser el sistema operativo que llevan el 78.7% de los teléfonos inteligentes vendidos en el primer cuarto de 2012. Seguido de lejos por BlackBerry, sistema operativo de la empresa RIM, el cual solo representa el 9.0% y Symbian con tan solo 6.2%. Todo esto según un estudio realizado por la firma Kantar Worldpanel[7].

Si comparamos estos datos con los obtenidos en el estudio de la firma Kantar Worldpanel en el mismo cuarto del año 2011, donde el porcentaje de móviles con sistema operativo Android era tan solo del 33.3%, concluimos que el crecimiento para el 2012 ha sido del 45.4%.

El secreto de este crecimiento tan rápido se debe a las características de las que presume el ecosistema Android. En primer lugar, como ya se ha explicado, Android está pensado para satisfacer los requerimientos de algunas de las empresas fabricantes de smartphones y tablets, aliadas bajo el nombre de Open Handle Alliance.

Estas empresas necesitan un ecosistema estándar, que sea ampliamente utilizado por todas las empresas fabricantes de dispositivos móviles que lo deseen. Y es que esto les permite obtener dos beneficios importantes:

- Las empresas fabricantes del hardware no necesitan embarcarse en el proceso de desarrollo de un sistema operativo propio, lo cual no es un proceso sencillo y conlleva consigo una inversión cuantiosa. Y estas empresas tienen disponible un sistema operativo creado y mantenido por la empresa Google, la cual hasta el momento se ha caracterizado por hacer buenas creaciones.

- En ecosistema móvil es muy importante el número de aplicaciones que estén disponibles, pues son estas aplicaciones las que dan utilidad al dispositivo.

  Teniendo un sistema operativo común para diferentes empresas, se consigue la compatibilidad con las mismas aplicaciones por parte de todos los dispositivos. Y, de esta forma, empresas sin tanto renombre puedan competir contra grandes empresas como son Nokia o Apple, las cuales por si solas y con sus propios sistema operativo móviles consiguen ventas muy importantes y mueven a un gran número de desarrolladores.

  En este sentido, Android es además un sistema muy abierto, lo que se traduce en muchas facilidades para los desarrolladores, factor que incrementa aún más el número de aplicaciones que podemos encontrar disponibles para el ecosistema.

Pero empresas importantes como Samsung, LG o HTC no se conforman con eso, pues estas quieren tener la opción de, de forma fácil y barata, personalizar la parte visual de Android, por tal de darle un toque único que se adapte a las características que estos quieren ofrecer a sus

usuarios. Android, al ser un sistema muy abierto, permite con facilidad esta personalización, tanto por parte de fabricantes de dispositivos móviles, como por parte de otros desarrolladores que quieran crear su propia interface.

Hay que decir, además, que el sistema operativo Android es de código abierto lo que aun facilita más la labor de las empresas que quieran implementar este sistema operativo en sus terminales, puesto que estos pueden acceder al código. Eso sí, para obtener la compatibilidad con Android, el fabricante no puede modificar el código completamente a su antojo, hay una serie de requisitos que deberá cumplir. En caso de no cumplirlos, no se pasará el test de compatibilidad y no se estará implementando el ecosistema Android.

Android es un sistema operativo gratuito, de esta forma los fabricantes no tienen que pagar por licencias a Google. Los beneficios que económicos que Google recibe son a través de la tienda de aplicaciones y de sus servicios de publicidad.

Resumiendo todo lo antes mencionado, el trabajo principal que deberá llevar a cabo el fabricante que quiera implementar el ecosistema, es el desarrollo de los drivers necesarios para que Android pueda comunicarse adecuadamente con el hardware del dispositivo, cualquier otro detalle queda en manos del fabricante si quiere personalizar más o menos el sistema. Por tanto, la inversión en software de estos fabricantes puede reducirse mucho gracias a Android.

Esta versatilidad de Android y la idea de tener un ecosistema común es la que ha hecho que un gran número de fabricantes añadan el ecosistema a sus dispositivos móviles y que, por tanto, este sistema operativo goce de gran popularidad.

El segundo punto importante que justifica el crecimiento tan rápido que ha experimentado el sistema operativo tiene que ver con el desarrollador de aplicaciones. Y es que este desarrollador tiene muchas ventajas a la hora de desarrollar para el sistema operativo Android.

Para empezar, publicar una aplicación en el Android Market Llamada es estos momentos “Google Play” (la tienda de aplicaciones de Google) es muy barato, únicamente necesitamos una licencia que tiene un coste de 25$ (unos 20€) para poder publicar aplicaciones durante un tiempo ilimitado. A parte de este pago, Google se lleva el 30% de las ganancias obtenidas con la aplicación, si es que esta es de pago. En caso de ser gratuita, no será necesario pagar nada más. Además, el entorno de desarrollo se puede montar tanto en Windows como en Linux e incluso Mac O.S.

Todas estas facilidades hacen que desarrollar para Android sea bastante más barato que hacerlo para otros ecosistemas móviles.

Las aplicaciones además, no sufren ningún tipo de control a la hora de publicarse. Desde el punto de vista de la libertad del desarrollador y de los usuarios esto es una ventaja, pues en el Android Market encontramos aplicaciones que en otras tiendas de aplicaciones de la competencia son

censuradas, simplemente por ir en contra de la política de la empresa o por tener contenidos para adultos. Pero también tiene su parte negativa, pues cualquier desarrollador puede publicar una aplicación malintencionada.

Para evitar que el usuario instale una aplicación con fines maliciosos sin darse cuenta, se ha desarrollado el sistema de permisos explicado con anterioridad y que recordamos a continuación. Cualquier aplicación que se instale en el dispositivo y quiera llevar a cabo una serie de accesos a la información externa a la propia aplicación, deberá solicitar al usuario los diferentes permisos necesarios para llevar a cabo estas acciones.

Así que, si por ejemplo, el usuario ve que un videojuego está pidiendo permiso para acceder a la información de los contactos o de las llamadas realizadas, podrá decidir no instalarlo, pues es sospechoso que este tipo de aplicación requiera tales permisos. En cambio, si instala el videojuego estará aceptando estos permisos y, por tanto, la aplicación podrá acceder a dicha información.

Como vemos se trata de un sistema que confía en la prudencia de los usuarios y, al mismo tiempo, en la buena fe de los desarrolladores. Por el momento, este sistema tan abierto está provocando algunas quejas por parte de algunos usuarios que lo consideran poco seguro.

Pero es que el termino abierto, en el sistema operativo Android, se extiende más allá de lo citado anteriormente, pues tampoco hay ninguna restricción que impida instalar aplicaciones externas al Android Market. Por tanto, las aplicaciones, se pueden instalar mediante ejecutables introducidos directamente dentro de la memoria del dispositivo o mediante otras tiendas de aplicaciones gestionadas libremente por otras empresas. Eso sí, la política de permisos se extiende más allá del Android Market, pues afecta a cualquier aplicación, que en caso de no obtener los permisos necesarios no podrá realizar las acciones deseada.

Android ha cosechado éxito gracias a la forma en la que se ha adaptado a lo que los fabricantes demandaban, así como gracias a la facilidad brindada a los desarrolladores para que puedan desarrollar sus aplicaciones para Android de forma barata y libre. Ecosistema libre, abierto y gratuito son los principales adjetivos que definen al sistema operativo Android.

2.3 Aplicaciones

2.3.1 Estructura de una aplicación

Las aplicaciones en Android están estructuradas en componentes, donde cada componente de la aplicación tiene un papel especifico dentro de la esta y existe por sí mismos, estos componentes a la vez puede que dependan uno de otro.

Al momento de desarrollar una aplicación para Android no tendremos una sola clase principal que lanzará las diferentes interfaces de usuario de la aplicación. Sino que tendremos múltiples componentes independientes comunicados entre sí a través del sistema operativo.

Estos componentes, también presentan algo más que un solo método principal que nos permita ejecutar el código de manera secuencial de la aplicación, implementan una serie de métodos que son convocados por el sistema operativo cada vez que se cumplan las condiciones necesarias para llamarlos. Por lo tanto las aplicaciones en Android se comportan de forma asíncrona y es el sistema operativo el que se encarga de gestionar estas llamadas según las peticiones del usuario.

El API de Android permite declarar cuatros tipos de componentes:

Services [8]

Un servicio es una tarea del sistema que se ejecuta en segundo planto o *Background*, tarea no tiene ninguna interfaz gráfica asociada, debido a que el usuario nunca interactúa de forma directa con un servicio. Este componente es el que se utiliza normalmente para realizar operaciones de larga duración de tal forma que el usuario no perciba que la operación se está llevando a cabo mientras se generan los resultados esperados o de realizar operaciones relacionadas con procesos remotos.

Activities [9]

Este componente representa una única pantalla de la aplicación y se encarga de relacionarla directamente con la interfaz grafica o vista con la que el usuario podrá interactuar. Entre las diversas cantidades de actividades que pueden formar una aplicación una de ella se marcará como la actividad principal o "*Main*", bajo esta actividad es que inicia la aplicación.

Las actividades pueden iniciar otras actividades con el objetivo de componer la aplicación de diversas pantallas o interfaces graficas con las que usuario pueda interactuar, tanto mostrando datos generados por la aplicación o facilitados por el usuario.

A modo de ejemplo podemos citar el reproductor multimedia el cual aun sin estar presente en la pantalla sigue ejecutando sus funciones.

Broadcast Receivers [9]

Este componente nos permite detectar y reaccionar ante ciertos mensajes o eventos globales generados por el sistema operativo como pueden ser; Bateria baja, SMS Recivido, Tarjeta SD insertada o cualquier otra aplicación del tipo boadcast, ya que cualquier aplicación puede generar mensajes o *intents*.

Se debe tener en cuenta que estos componentes son representados por clases Java, lo que da a entender que cuando la aplicación se ejecuta, se crean instancias de estos, pudiéndose crear más de una instancia del mismo componente y dichas instancias son independientes.

Intents [9]

Un *intent* es el elemento básico de comunicación entre los distintos componentes Android que hemos descrito anteriormente. Se pueden entender como los *mensajes* o *peticiones* que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un *intent* se puede mostrar una *actividad* desde cualquier otra, iniciar un servicio, enviar un mensaje *broadcast*, iniciar otra aplicación, etc.

Content Provider [10]

Bajo la utilización de este componente es posible acceder a datos que estemos necesitando en nuestra aplicación desde las diferentes fuentes de datos que nos proporciona Android. Según como sea configurado el Proveedor de contenido limitaremos el contenido a unas aplicaciones muy concretas o no. También puede almacenar datos en los ficheros del sistema, en la base de datos SQLite de la aplicación, una página Web o en cualquier otro lugar de almacenamiento.

2.3.2 *AndroidManifest*

Cada aplicación Android que se desarrolle debe de tener un archivo con el nombre AndroidManifest.xml [11]. Este fichero contiene información sensible sobre la aplicación que el sistema operativo debe conocer. Esta información es indispensable para que la aplicación pueda ejecutarse.

Entres las informaciones que maneja este archivo podemos encontrar:

- Contiene el nombre del paquete de desarrollo de la aplicación, el cual se utiliza como identificador único de la aplicación dentro del sistema.

- Define cada uno de los componentes de la aplicación, las *activities*, *services*, *brodcast recievers* y los *content provider* los cuales componen una aplicación. También hace referencia a las clases que implementan estos componentes y da a conocer sus propiedades.

- Determina que procesos son los que se encargan de acoger a los componentes implementados por la aplicación.

- Indica los tipos de permisos que debe tener la aplicación a la hora de querer acceder a partes protegidas de la API e interactuar con otras aplicaciones.

- Se declara el nivel mínimo y la versión para que fue creada de la API que el terminal Android debe tener instalado.

2.3.3 Las Actividades

Debido a que las actividades son el componente más importante en una aplicación [12], a continuación se detallara el funcionamiento de dicho componente.

Cualquier aplicación debe de tener por lo menos una actividad. Hablamos de la actividad principal, la cual se ejecutará al momento del usuario indicar que desea ejecutar dicha aplicación.

Para crear un componente de tipo actividad, es necesario crear una clase en Java que herede la clase *Activity*. Esta clase se le deberá asociar una interfaz o vista que la actividad le mostrará al usuario. Android nos facilita una serie de herramientas para definir estas interfaces utilizando el lenguaje XML, los que nos da la posibilidad de separar la parte visual de la actividad de la parte lógica de esta, siendo la parte lógica la que respondería a una entrada por parte del usuario y realizaría ciertas operaciones para ofrecer una respuesta. Esta lógica se implementa a través del lenguaje Java.

La definición de una interface para una actividad en Android se lleva a cabo mediante la construcción de un árbol de elementos gráficos llamados *views*, figura 2.3. Las views pueden estar prefabricadas o desarrolladas por el propio desarrollador de la aplicación, según las necesidades de este.

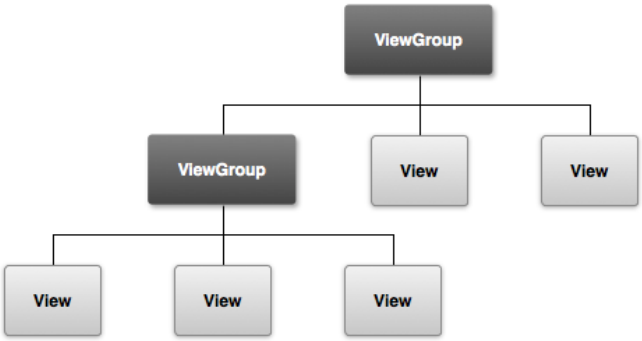

Figura 2.3 Árbol jerárquico View

Una view es una estructura de datos que mantiene parámetros y contenidos para gestionar la interacción del usuario con un rectángulo específico [13]. Este rectángulo es un fragmento del área de la pantalla ocupado por la actividad cuya interface está compuesta por la view en cuestión.

Al momento de visualizar una actividad, esta se mostrará en un área específica de la pantalla, más grande o más pequeño según si la estamos visualizando a pantalla completa o no. La construcción de una interface consiste en dividir dicha área en rectángulos, cada uno gestionado a través de una *view* diferente.

Existen dos tipos de *views*: los *ViewGroup*, que son *views* que dentro pueden tener otras *views* de cualquier tipo y dotan a estas *views* interiores de una serie de propiedades concretas; y los *widgets*, que son *views* raíz, las cuales llevan a cabo algún tipo de interacción con el usuario, independiente del resto de *views*, dentro del rectángulo que se les asigna.

Ejemplos de *ViewGroup* son las layouts, unas *views* que condicionan la forma en que el área ocupada por el *ViewGroup* se reparte entre sus *views* internas. La repartición puede ser dividiendo el área horizontalmente, verticalmente, etc. dependiendo del tipo de *layout* escogida.

En cambio, ejemplos de *widgets* son: el botón, la caja para rellenar con texto, la lista de texto, entre otras. La mayoría de *widgets* necesarios ya vienen prefabricados en la API de Android y ya traen implementada la interacción con el usuario. Por tanto, en caso de utilizarse un *widget* prefabricado, el desarrollador únicamente deberá leer los datos que el *widget* recoge del usuario y llevar a cabo con ellos el propósito que desee.

Para poder leer estos datos, en lugar de llevar a cabo una comprobación constante, normalmente se deben definir una serie de operaciones que, correctamente configuradas, son llamadas por el sistema operativo cuando una de las *views* tiene nueva información que presentar. Nuevamente vemos como el funcionamiento de los componentes de una aplicación en Android es asíncrono y es el sistema operativo el que nos avisa de los sucesos cuando ocurren. En concreto, en una actividad, con el mecanismo asíncrono evitando bloquear el hilo donde se está ejecutando dicha actividad con bucles largos, lo cual provocaría que no se pudiera llevar a cabo una interacción con el usuario.

2.3.4 Ciclo de vida de una actividad

Una actividad en Android funciona de forma asíncrona y el hilo donde se ejecuta esta actividad debe quedar libre por tal de que los eventos que activa el usuario se reciban con rapidez.

Por este motivo, en la clase *Activity* se declaran una serie de operaciones, las cuales deberemos sobre-escribir en la actividad que creemos, pues a través de ellas llevaremos a cabo las acciones que queremos que dicha actividad haga. Estos métodos son llamadas por el sistema operativo cuando se dan una serie de sucesos concretos, son las operaciones a través de las cuales gestionamos el ciclo de vida de la actividad.

A continuación se describen los métodos a través de las cuales se gestiona el ciclo de vida de la actividad [12]:

***onCreate*():** Operación que llama el sistema operativo a la hora de crear la instancia de la actividad.

Esta operación es muy importante, pues es donde tenemos que introducir la jerarquía de *views* que define la interface de la actividad. Esto se hace mediante la llamada a *setContentView*(), independientemente de si cargamos dicha jerarquía de un fichero XML (lo más aconsejable) o de si la definimos en el propio código Java. Además, también deberemos inicializar los datos y llevar a cabo las operaciones cuyos resultados se mantengan estáticos durante toda la ejecución de la actividad.

***onStart*():** Esta operación será llamada por el sistema operativo cuando la actividad vaya a pasar a ser visible para el usuario, antes de hacerlo.

En ella deberemos llevar a cabo los preparativos para una visualización concreta de la actividad. Preparativos que se deberán hacer cada vez que se visualice la actividad de nuevo, pues en caso contrario se habrían llevado a cabo en *onCreate*().

***onResume*():** Esta operación la llama el sistema operativo justo después de que la actividad pase a ser visible para el usuario y hasta que no se acabe su ejecución la interacción con el usuario no podrá llevarse a cabo. Se suele utilizar para activar animaciones o efectos que deben empezar justo cuando el usuario ya está viendo la actividad.

***onPause*():** Otra de las operaciones más importantes en el ciclo de vida de una actividad. Esta operación es llamada por el sistema operativo cuando la actividad va a pausarse, pues otra actividad va a pasar a ejecutarse en primer plano. Se trata de la última operación que el sistema operativo Android nos asegura que será llamada. Y es que, una vez ejecutada esta operación, por falta de memoria principal el sistema operativo puede matar la actividad antes o después de ejecutar *onStop*(). Por este motivo, en *onPause*() deberemos hacer persistentes los datos que aún no hayamos guardado.

Además, en esta operación también se suelen pausar animaciones u operaciones que se estén ejecutando en otro thread y consuman mucha CPU, pues cuando la actividad se pausa probablemente estas operaciones ya no es necesario que se estén ejecutando.

En cualquier caso, las tareas de *onPause*() se deben llevar a cabo rápidamente, pues hasta que no acabe su ejecución la actividad no podrá dejar de ser visible y, por tanto, la nueva actividad que vaya a ser presentada no podrá visualizarse.

***onStop*():** El sistema operativo llama a esta operación cuando la actividad acaba de ser retirada de la vista del usuario. Aquí se suelen llevar a cabo acciones que liberan los recursos del dispositivo y que mientras la actividad estaba visible no se podían llevar a cabo, ya sea porque habrían tardado mucho en realizarse o porque el usuario las habría visto.

El sistema operativo puede matar la actividad antes, durante o después de esta operación si necesita liberar memoria principal.

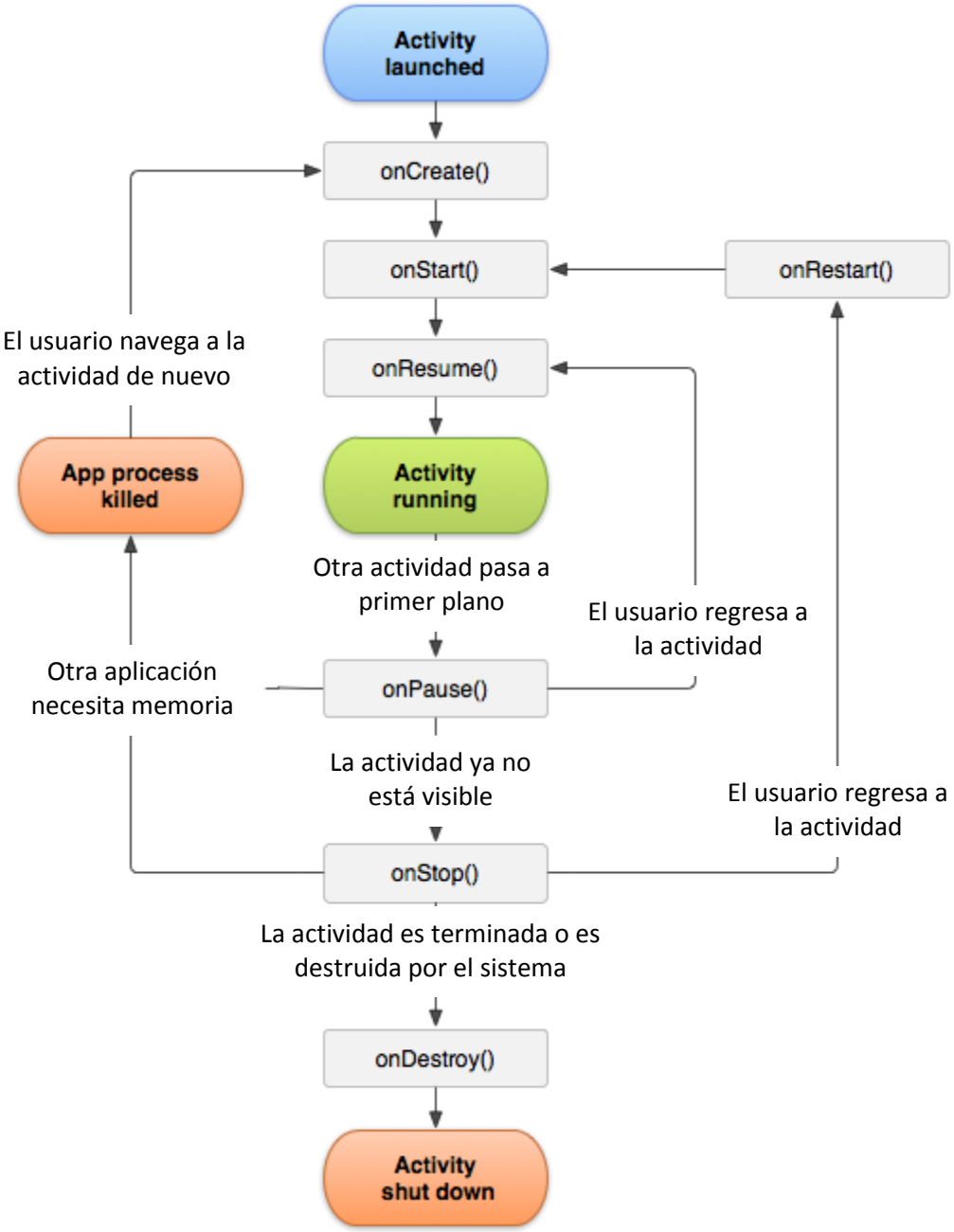

Figura 1.4 Esquema del Ciclo de Vida de una Aplicación Android

***onRestart*():** Esta operación se llama cuando la actividad, que permanece invisible, va a volver a visualizarse. En ella se pueden llevar a cabo preparativos que únicamente sea necesario realizar cuando la actividad pasa a ser visible después de una pausa anterior. Ver Figura 2.4 para más detalles acerca de las operaciones llamadas posteriormente a esta.

Una vez llamada la operación *onRestart*(), la actividad nuevamente no podrá ser eliminada por el sistema operativo hasta después de que se vuelva a llamar a *onPause*(), pues el sistema operativo da la prioridad más alta a la actividad visible en cada momento.

onDestroy(): Esta operación será llamada por el sistema operativo cuando la instancia de la actividad en cuestión vaya a ser eliminada, antes de llevar a cabo la eliminación. En ella se deben realizar las acciones de liberación de recursos necesarias.

Esta operación solo será llamada si la actividad va a ser eliminada por indicación explicita del desarrollador de la aplicación, a causa de que el usuario haya presionado el botón Back o debido a una necesidad de recursos leve por parte del sistema operativo Pero si, por el contrario, el sistema operativo necesita memoria rápidamente, este eliminará el proceso entero donde se esté ejecutando la actividad sin llamar a *onDestroy*(), aunque tampoco hará falta, pues al eliminar el proceso se liberarán todos los recursos automáticamente.

Y hasta aquí las operaciones a implementar para gestionar el ciclo de vida de una actividad. Nótese que no todas tienen por qué redefinirse en la subclase de *Activity* que creemos, solo las operaciones en las que nos sea necesario llevar a cabo algunas acciones. Para el resto de operaciones dejaremos el comportamiento por defecto implementado en la superclase *Activity*.

2.3.5 Ejecución de una Aplicación

En Android, las aplicaciones se ejecutan cada una en su propia instancia de la máquina Dalvik. La máquina Dalvik es una máquina virtual y su trabajo es interpretar, en tiempo real, el código en el que están escritas las aplicaciones para Android, transformándolo en instrucciones que el procesador del dispositivo entienda. La decisión de utilizar una máquina virtual que en tiempo real interpreta el código se debe a la necesidad de que Android sea un ecosistema que se pueda utilizar en múltiples dispositivos diferentes, con arquitectura diferente.

Para conseguir un código multiplataforma, en el desarrollo de aplicaciones para Android se utiliza el lenguaje Java, que también es un lenguaje interpretado durante la ejecución. Gracias a la utilización de este lenguaje el hecho de que una aplicación funcione en un determinado dispositivo dependerá únicamente de que la máquina virtual, que interpreta el código en tiempo real y que viene incluida con el paquete de software Android, este adaptada a la arquitectura del dispositivo. Evitamos así tener que compilar el código para cada arquitectura diferente y tan solo

debemos compilarlo una única vez para que este se transforme en un código compatible con la máquina Dalvik.

Esta máquina virtual esta especialmente optimizada para que se puedan ejecutar varias instancias de ella al mismo tiempo. Y es que Android es un sistema operativo multitarea y varias aplicaciones pueden estar ejecutándose a la vez. En este caso, cada una de ellas utilizará una instancia diferente de la máquina Dalvik. Además, entre diferentes instancias de la máquina Dalvik no puede haber una comunicación directa, pues así se consigue mayor seguridad en el ecosistema.

A modo de ejemplo, suponiendo una instancia que está ocupada por una aplicación la cual hace la función de cliente de correo electrónico y además esta aplicación tiene dos instancias de componentes en ejecución: un servicio que va comprobando si se han recibido nuevos mensajes para alertar al usuario y una actividad que presenta al usuario la pantalla de redacción de un correo electrónico. Esta última actividad, la ha lanzado el navegador de Internet que se está ejecutando en la segunda instancia de la máquina Dalvik.

Por tanto, podemos concluir que dentro de cada instancia de la máquina Dalvik se encuentran las diferentes instancias de los componentes de una aplicación determinada que se han creado en un momento dado.

Por defecto y a menos que queramos lo contrario, todos los componentes de una instancia de la máquina Dalvik se ejecutan en el mismo proceso.

Y en lo que respecta a los hilos o *thread* de ejecución, en principio, todas las instancias de componente se ejecutan también en un único hilo, llamado *UI Thread*. El *UI Thread* tiene prioridad ante el resto de hilos que pueden crearse en un proceso de una instancia de la máquina Dalvik, pues se trata del hile que recibe las peticiones del usuario y del sistema operativo, así que contra más rápido se reaccione a estas peticiones más positiva será la experiencia de usuario.

Por este motivo debemos evitar que el *UI Thread* realice demasiado trabajo y, para conseguirlo, cuando una petición del sistema operativo requiera una serie de operaciones costosas deberemos crear otro hilo (también llamado *Worker Thread*) donde llevarlas a cabo. De igual forma, si creamos un servicio, lo aconsejable es que este se ejecute en su propio hilo, en lugar de en el UI Thread.

Capítulo 3.

Diseño de la aplicación

Basándose en la gestión y ventas de contenidos se presenta el diseño de la aplicación detallando sus funcionalidades internas y externas.

3.1 Presentación de la aplicación

El proyecto se basará en el desarrollo de una aplicación para el sistema operativo Android bajo el nombre oliva, la cual consiste en la gestión y venta de contenidos multimedia alojados en un servidor remoto como son, Imágenes, Documentos, Audio y Video para kioscos ubicados en lugares específicos como podrían ser lugares de interés turísticos.

A modo de ejemplo esta aplicación podría ser fácilmente utilizada en museos donde al momentos de visitarlos este le ofrezca a los visitantes la ventas de ciertos contenidos multimedia como bien podría ser unas de sus presentaciones del momento (video), la impresión o descarga de alguno de los cuadros famosos o que le gusten al visitante (imágenes), en vez de utilizar folletos impresos, ya estos estén presente en la aplicación y se reduce el uso de papel (documentos) o incluso la descarga de algún tipo de audio que represente algo atractivo para el visitante del museo.

Pero no tan solo sería aplicable a lugares donde se tenga que pagar por el contenido, podríamos poner el caso de un punto turístico donde se quieras dar a conocer a los turista lo que verán o

como sería la mejor forma de disfrutar el lugar que están visitando. Con tan solo descargarse la aplicación podrían ver y/o descargar el contenido que se les quiera ofrecer.

Como base principal las cosas antes mencionadas fueron las que dieron paso a las investigaciones acerca de Android para conocer todo lo que este sistema operativo nos puede ofrecer y así desarrollar una aplicación que pudiese realizar las tareas antes presentadas.

3.2 Funcionalidades Internas

El usuario a la hora de utilizar la aplicación podrá:

- Visualizar todo el contenido posible, Imágenes, Documentos, Audio y Video y podrá tener una vista previa de estos antes de proceder a realizar algún tipo de pago. Por ejemplo si quiere descargar una imagen esté podrá ver una imagen con una resolución baja la cual no se apreciar en el terminal. Si es caso de que el contenido sea audio o video este podrá reproducir una porción del audio o del video.

- En cualquier momento que desee podrá saber la de archivos por tipo de contenido que contenga la aplicación

- Después de que el usuario haya interactuado con el contenido este podrá elegir, para los contenidos que apliquen, en sí descargarlos o imprimirlos

- A la hora de realizar el pago, el usuario tendrá la posibilidad de realizar el pago a través de PayPal.

3.3 Funcionalidades externas.

Cuando nos referimos a funcionalidades externas de lo que hablamos es del servidor de datos. Para que la aplicación sea funcional necesita descargar el contenido que presentará desde un servidor remoto, este servidor es el que se encargará de gestionar la creación de un archivo XML que tendrá la información del contenido multimedia del servidor, los precios para las acciones imprimir o descargar y también será el que gestione la impresión.

Capítulo 4.

Implementación de la aplicación

A continuación se explicará cómo se ha creado la aplicación, las diferentes tecnologías que han sido empleadas y la forma y los procesos que se siguen para llevar a cabo la prueba de conceptos.

4.1 Objetivos de la implementación

Siendo el objetivo principal de la aplicación ofrecer una interfaz cómoda para el usuario donde este pueda fácilmente interactuar con el contenido disponible, pre visualizarlo y comprarlo con una ayuda nula. El usuario antes de pagar por el contenido tendrá la opción a elegir si desea descargarlo o imprimirlo para los casos de ser documentos o fotos.

Para poder cumplir con el objetivo principal fue necesario plantearse unos cuantos objetivos a tener en cuenta a la hora de ir desarrollando la aplicación. Entre estos objetivos se encuentran:

Interfaz cambiante: Al momento de cargar el contenido disponible existen iconos que acompaña al contenido, los cuales para poder ofrecer un entorno más agradable deben ser modificado sin tener que recompilar la aplicación.

Sencillez y Robustez: Este es uno de los objetivos más críticos y problemáticos a la vez, cuando estás dos palabras se unen, sencillez y robustez significan que encada momento la experiencia del usuario debe ser la mejor pero sin que el software deje de ser eficiente.

Escalabilidad: El código creado debe ser "fácilmente modificado" con el fin de poder agregar nueva implantaciones sin tener que hacer cambios masivos en este permitiendo así modificaciones en poco tiempo.

Contenido Remoto: Todo el contenido que estará disponible en la aplicación estará ubicado en un servidor, lo cual permitirá crear una aplicación que ocupe poco espació y de contenido variable.

Utilización de estándares de comunicación abiertos: Algo a tener en cuenta es que el desarrollo del servidor que utiliza la aplicación no ha sido desarrollado sino que se ha creado un servido tonto para realizar las pruebas donde nos comunicamos con este vía HTTP y recibimos respuesta en el formato XML.

Métodos de pagos confiables: A la hora de los usuarios realizar un pago a través de un equipo móvil se ve casi obligado, por términos de confianza a realizar sus pago a través de plataformas de pagos conocidas como son PayPal y Google Wallet.

4.2 Partes específicas de Android tratadas

Los conceptos de Android que se han investigado y tratado para poder realizar la aplicación son;

- Manipulación básica de XML para poder crear interfaces graficas con los elementos que nos facilita Android.
- Creación y llamada de actividades y el ciclo de vida de estas.
- Implementación de las clases *AsyncTask, BaseAdapter*.
- Implementación de Hilos
- Uso de librerías externar como por ejemplo la de PayPal para la implementación de pagos.
- Implementación de los valores "resources".
- Buen entendimiento de los componentes que conforman un proyecto Android, como son el directorio principal, el directorio layouts, el archivo Manifiest, Etc.
- Manejo e implementación de notificaciones

4.3 Arquitectura de servicio

Figuea 4.1 Sistema Completo

4.3.1 Terminal Android:

Durante todo el proceso de desarrollo de la aplicación se estuvo utilizando una Tablet de 7 pulgadas. La Tablet dispone del sistema Operativo Android 2.3.3 y reproductor de audio y video que ya viene pre instalado con el sistema operativo y un aplicación para visualizar documentos del tipo PDF, PPT, PPS, WORD, XLS.

4.3.2 Servidor:

El servidor por su parte no es totalmente funcional sino que se asumió como una caja negra y la creación del XML se realizó de forma manual y se creó una respuesta falsa a la hora de la impresión. EL servidor está implementado bajo Apache TomCat y el uso de servlet.

4.4 Estructura de la Aplicación

En este apartado se estará presentando la estructura interna de la aplicación realiza en base a los conocimientos adquiridos sobre Android.

4.4.1 Directorio “src”

En este directorio se encuentran los diferentes paquetes Java que componen la aplicación, a continuación se explicará el contenido de cada uno y así poder encontrar de una forma eficiente el código que pertenece a las diferentes funcionalidades.

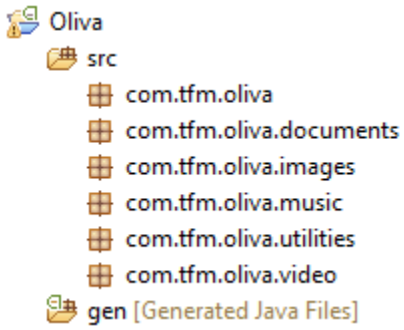

Figura 4.2 Directorio "src"

Las funcionalidades de los paquetes se presentan a continuación, dando una breve explicación de las clases que componen cada paquete:

4.4.2 com.tfm.oliva

Este paquete contienes las clases principales, en Android llamadas activity, entre ellas se encuentran:

LauncherActivity*:* Es la actividad de inicio de la aplicación esta tendrá la función de presentar una imagen de bienvenida al usuario mientras se realiza la primera conexión con el servidor y se descargar el archivo XML con la información de los contenidos a utilizar por la aplicación. Éste archivo XML se le envía a la actividad *OlivaMainActivity.*

OlivaMainActivity*:* Es la actividad principal del programa donde su función principal consiste en cargar la interfaz principal de la aplicación que está compuesta por un TabView donde se presentarán los diferentes tipos de contenidos de la aplicación e inicializar la librería de pago de PayPal.

Al momento de iniciar el TabView es necesario llamar a las actividades que lo componen siendo en este caso las actividades; *ImagenActivity, MusicActivity, DocumentsActivity, VideoActivity*. Durante la llamada se les envía a cada actividad el archivo XML para Después estas procesarlos de forma individual.

HomeActivity: Es la actividad que encarga de mostrarle al usuario un resumen del contenido disponible a hacer ofertado

ImagenActivity: Es la actividad que encarga de cargar las imágenes disponibles en el servidor y hereda la clase *ExpandableListActivity* ya que se estará utilizando una lista expandible (*ExpandableList)* para presentarles las imágenes al usuario. Esta actividad utiliza las clases alojadas en el paquete *com.tfm.oliva.images*, que se estarán explicando más adelante.

MusicActivit Es la actividad que encarga de cargar los archivos de audios disponibles en el servidor y hereda la clase *ExpandableListActivity* ya que se estará utilizando una lista expandible (*ExpandableList)* para presentarle los archivos de audio al usuario. Esta actividad utiliza las clases alojadas en el paquete *com.tfm.oliva.music*, que se estarán explicando más adelante.

DocumentsActivity: Es la actividad que se encarga de cargar los documentos disponibles en el servidor y hereda la clase *ExpandableListActivity* ya que se estará utilizando una lista expandible (*ExpandableList)* para presentarles los documentos al usuario. Esta actividad utiliza las clases alojadas en el paquete *com.tfm.oliva.documents*, que se estarán explicando más adelante.

PayActivity: Es la actividad encargada de recibir la información del contenido que se quiero adquirir y gestionar el pago a través de PayPal.

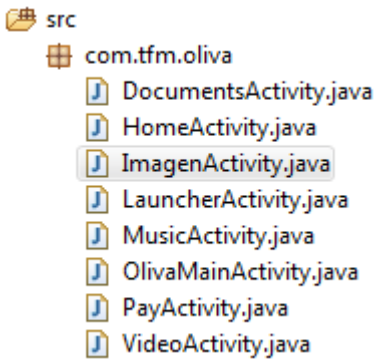

Figura 4.3 Paquete com.tfm.oliva

4.4.3 com.tfm.oliva.documents

Este paquete contiene las clases auxiliares que utiliza *DocumentsActivity* para poder realizar sus funciones, entre estas clases están:

DocumentsAdapter: Esta clase se extiende de la clase tipo adaptador *BaseExpandableListAdapte*, ya que como los documentos que se van a utilizar serán mostrado en una Lista Expansible, esta clase trae consigo todo lo necesario para poder mostrar la información deseada en las listas Expansibles. Los datos que serán presentados se obtienen de la clase *DocumentListLoading.*

Por otro lado esté adaptador se auxilia de algunas de las clases del paquete *com.tfm.oliva.utilities* para poder descargar las imágenes correspondiente que se presentarán en la lista.

Las lista expansible se pueden dividir en dos, en grupos y en hijos, los grupos contienen a los hijos y por lo tanto estas listas van asociada a dos interfaces graficas una para los hijos y otras para los grupos.

DocumentListLoadin: Esta clase se extiende de la clase AsyncTask, la cual permite realizar actividades en un hilo diferente al hilo principal de ejecución y así no dar la sensación de que la aplicación se ha detenido. El uso de la AsyncTaskes es debido a que se procesarán operaciones que demorarán tiempo como es la llamada *DocumentsAdapter* para rellenar la lista expandible Después de que *DocumentListLoadin* procese el archivo XML.

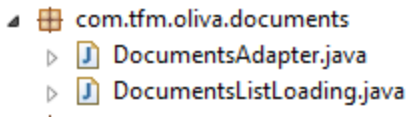

Figura 4.4 Paquete com.tfm.documents

4.4.4 com.tfm.oliva.images

Este paquete contiene las clases auxiliares que utiliza *ImagenActivity* para poder realizar sus funciones, entre estas clases están:

ImagesListLoading: Esta clase se extiende de la clase AsyncTask, la cual permite realizar actividades en un hilo diferente al hilo principal de ejecución y así no dar la sensación de que la aplicación se ha detenido. El uso de la AsyncTaskes es debido a que se procesarán operaciones que demorarán tiempo como es la llamada *ImagesAdapter* para rellenar la lista expandible Después de que *ImagesListLoading* procese el archivo XML.

ImageLoader: Esta clase se utiliza para descargar las imágenes thumbnail desde el servidor, la peculiaridad de esta clase es que funciona como cargador de imágenes del tipo perezoso.

ImagesAdapter: Esta clase se extiende de la clase tipo adaptador *BaseExpandableListAdapter*, ya que como los imágenes que se van a mostrar serán cargados en una Lista Expansible, esta clase trae consigo todo lo necesario para poder mostrar la información deseada en las listas Expansibles. Los datos que serán presentados se obtienen de la clase *ImagesListLoading.*

ImagesFullActivity: Esta actividad se llama desde la actividad ImagenActivity al momento de que se quiera visualizar un tamaño mayor que la imagen de muestra que tiene La lista Expandible la actividad ImagenActivity. Esta actividad implementa un ImagenSwitcher y una galería de imágenes.

ImagesGalleryAdapter: Esta clase se extiende de la clase tipo adaptador *BaseAdapter*, ya que como los imágenes que se van a mostrar serán cargados en una galería, esta clase trae consigo todo lo necesario para poder mostrar la información deseada en la galería. Los datos que serán presentados se obtienen de la clase *ImagesListLoading.*

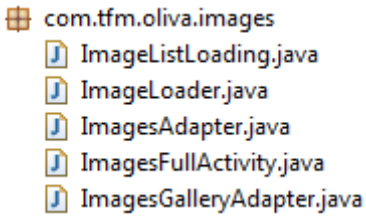

Figura 4.5 Paquete com.tfm.images

4.4.5 com.tfm.oliva.music

Este paquete contiene las clases auxiliares que utiliza *MusicActivity* para poder realizar sus funciones, entre estas clases están:

MusicAdapter: Esta clase se extiende de la clase tipo adaptador *BaseExpandableListAdapter*, ya que como los imágenes que se van a mostrar serán cargados en una Lista Expansible, esta clase trae consigo todo lo necesario para poder mostrar la información deseada en las listas Expansibles. Los datos que serán presentados se obtienen de la clase *MusicListLoading*

MusicListLoading: Esta clase se extiende de la clase *AsyncTask*, la cual permite realizar actividades en un hilo diferente al hilo principal de ejecución y así no dar la sensación de que la aplicación se ha detenido. El uso de la *AsyncTask* es debido a que se procesarán operaciones que demorarán tiempo como es la llamada *MusicAdapter* para rellenar la lista expandible Después de que *ImagesListLoading* procese el archivo XML.

MusicPlayActivity: Esta actividad se llama desde la actividad *MusicActivity* al momento de que se quiera reproducir el archivo audio de muestra que tiene La lista Expandible en la actividad *MusicActivity*. Esta actividad implementa MediaPlayer para realizar su función.

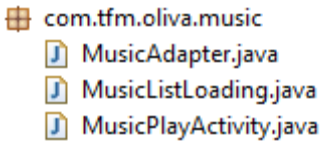

Figura 4.6 Paquete com.tfm.music

4.4.6 com.tfm.oliva.video

VideoListLoading: Esta clase se extiende de la clase *AsyncTask*, la cual permite realizar actividades en un hilo diferente al hilo principal de ejecución y así no dar la sensación de que la aplicación se ha detenido. El uso de la *AsyncTask* es debido a que se procesarán operaciones que demorarán tiempo como es la llamada *VideoAdapter* para rellenar la lista expandible Después de que *VideoListLoading* procese el archivo XML.

VideoAdapter: Esta clase se extiende de la clase tipo adaptador *BaseExpandableListAdapter*, ya que como los videos que se van a mostrar serán cargados en una Lista Expansible, esta clase trae consigo todo lo necesario para poder mostrar la información deseada en las listas Expansibles. Los datos que serán presentados se obtienen de la clase *VideoListLoading.*

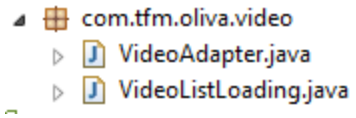

Figura 4.7 Paquete com.ftm.video

4.4.7 com.tfm.oliva.utilities

Este paquete está compuesto por un grupo de clases que sirven de auxiliares para alguna de las actividades principales entre sus clases tenemos:

DownloadTask*:* Es la clase que se encarga de realizar la descarga Después de que se paga el contenido conectándose al servidor vía HTTP.

FileCahe: Es la clase que genera las rutas en la memoria del terminal donde se guardaran los archivos temporales de la aplicación.

Helper: Esta clase contienes unos métodos para la verificación de la conectividad y la conectividad al servidor para verificar la disponibilidad para imprimir.

MemoriaCache: Esta clase le sirve de auxiliar a la clase *Imageloader* para mantener los thumbanail en memoria.

NotificationHelper: Esta clase se encarga de presentar una notificación en la *barra de estado del teléfono.*

Utils: Esta clase ayuda a la clase *ImageLoader* al momento de esta descargar la imagen desde el servidor y alojarla en la memoria del terminal.

XMLParser: Esta clase se utiliza para descargar el archivo XML desde el servidor y a la vez contiene métodos para realizar la validación del mismo.

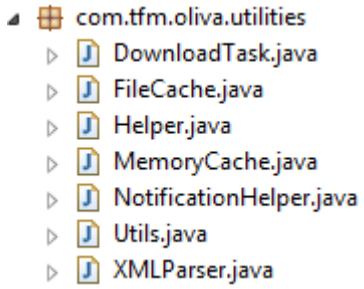

Figura 4.8 Paquete com.tfm.utilities

4.4.8 Directorio “res”

En este directorio es donde se almacenan los recursos externos, como imágenes, strings, etcs, los cuales utiliza la aplicación para su funcionamiento. En el directorio *drawable* se almacenan las imágenes y en el directorio *layout* se encuentran los XML que definen las vistas en la aplicación.

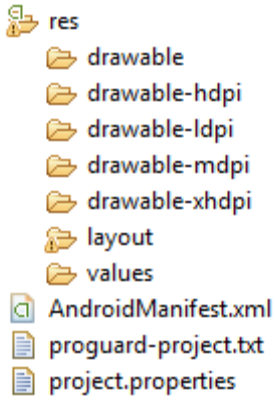

Figura 4.9 Directorio “res” de la aplicación

4.5 Implementación de actividades

Unas de las características que acompañan a Android a la hora de querer implementar una interfaz de usuario estas interfaces deben ir asociadas a una actividad ya que las actividades son las que gestionan los eventos que se producen sobre las interfaces.

En lo que respecta a este apartado estaremos viendo algunas de las actividades que forman parte de la aplicación debido a que algunas se parecen bastante en su implementación con tan solo unos ligeros cambios, esto se pudo apreciar en al apartado anterior donde se explicaban las clases pertenecientes a cada paquete.

4.5.1 *OlivanMainActivity*

Esta actividad se encuentra en el paquete *com.tfm.oliva* al igual que las otras actividades principales. Todas las actividades deben estar registradas en el archivo *AndroidManifest.xml* el cual se encuentra en el directorio del proyecto.

Como se ha mencionado en el punto anterior cada actividad debe ir asociada a una interfaz gráfica para el usuario, en este caso, la actividad *OlivanMainActivity* se encargará de manejar los

eventos producidos en la interfaz principal (main) declara en la carpeta layout. Esta actividad se extiende o hereda *TabActivity* en vez de *Activity* ya que la interfaz principal es del tipo TabHost

La idea de utilizar un *TabHost* viene debido a que se quería que el usuario no tuviese que utilizar ningún menú para ir entre las diferentes categorías de datos que contiene la aplicación. Los *TabHost* contienen tabs a los cuales les puede asociar una actividad y por lo tanto se le asociaron las actividades que manejan las diferentes categorías de la aplicación. En vez de utilizar nombre para cada tabs se implementaron iconos ya estandarizados para facilitar la interacción del usuario.

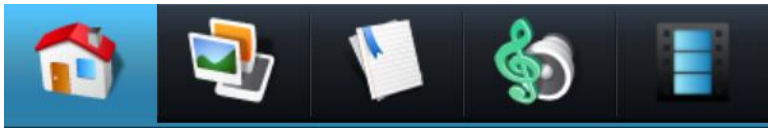

Figura 4.10 TabHost actividad principal con 5 tabs

Esta actividad además de la implementación del *TabHost* es la encargada de enviarle el archivo XML previamente descargado por la actividad *LauncherActivity*, a las diferentes actividades a la hora de iniciar cada actividad.

```
// Implementación del tab Video
TabSpec videoSpec = tabHost.newTabSpec(VIDEO_SPEC);
videoSpec.setIndicator("",
        getResources().getDrawable(R.drawable.ic_tab_video));
//Creación del intent asociada a la actividad a iniciar cargándole
//el  contenido del archivo XML
Intent videoIntent = new Intent(this, VideoActivity.class);
videoIntent.putExtra("xml", Ilauncher.getStringExtra("xml"));
videoSpec.setContent(videoIntent);
```

Figura 4.11 creación del tab Video asociado al TabHost

Otra función que tiene esta actividad es la inicialización de la librería de pago de PayPay. La librería se inicia en esta actividad debido a que demora un tiempo el proceso y para evitar que a la hora de realizar el pago se esté iniciando solo se inicia una vez en esta actividad y a la hora de hacer el pago se puede realizar más rápido.

4.5.2 *ImagenActivity, DocumentsActivity, MusicActivity, VideoActivity*

Este conjunto de actividades tienten la característica de con casi idénticas en el código que implementan, la razón por la que son casi idénticas se debe a que se quiso al hora de desarrollar la aplicación que los usuarios pudiesen visualizar el contenido en listas desplegables.

Figura 4.12 Representación de las Listas Expandible

Por esta razón y debido a que la interfaz asociada a estas actividades implementa listas desplegables *(*ExpandableListView*)* todas estas actividades en vez de heredar de la clase *activity* heredan de la clase *ExpandableListActivity*.

Al momento de gestionar el evento que se produce por pulsar a un hijo de las listas expansible dependiendo de la actividad está realizará una tarea diferente debido a las diferentes formas de visualizar o reproducir el contenido.

```
@Override
public boolean onChildClick(ExpandableListView parent, View v,
        int groupPosition, int childPosition, long id) {

    //Se crea el intent para iniciar la actividad MusicPlayActivity
    //Al intente se le pasan la posición del hijo seleccionado y el
    //grupo al que pertenece
    Intent i = new Intent(getApplicationContext(), MusicPlayActivity.class);
    i.putExtra("groupPosition", groupPosition);
    i.putExtra("childPosition", childPosition);

    startActivity(i);
```

Figura 4.13 Método que gestiona el evento producido al dar click sobre hijo de un grupo

Solo para las actividades *MusicActivity* e *ImagenActivit*y al momento de pulsar sobre uno de los hijos se utiliza una actividad implementada para visualizar el contenido y las actividades Las actividades *VideoActivity* ***y*** *DocumentsActivity* llaman a programas externos, aplicaciones ya instaladas en el terminal para visualizar su contenido.

Otros eventos manejados por estas actividades son el *onBackPressed()* y el *onCreateOptionsMenu(menu menu).* El primero de los métodos mencionados es el que se llama al momento de pulsar la tecla "atrás" del terminal Android, el cual ha sido sobre escrito con el fin de preguntar si realmente quiere cerrar la aplicación o no. Y el segundo le presenta un menú al usuario al momento de pulsar la tecla "menú" del terminal Android para en el cual podrá elegir si salir de la aplicación o mantenerse en esta.

Estas actividades, también, debido a su gran similitud todas se auxilian de dos clases las cuales para cada actividad tienen nombre diferentes pero su comportamiento es muy parecido.

Dependiendo de la actividad estas clases son:

ImagenActivity:

- *ImagesListLoading*
- *ImagesAdapter*

DocumentsActivity:

- *DocumentsListLoading*
- *DocumentsAdapter*

MusicActivity:

- *MusicListLoading*
- *MusicAdapter*

VideoActivity:

- *VideoListLoading*
- *VideoAdapter*

La razón por la cual no simplemente se utilizo las mismas clases para todas las actividades se debe a que existen ciertas variaciones en los componentes de las vistas que acompañan a las actividades y en los nombres de los campos del archivo XML a la hora de extraer los datos.

4.5.2.1 ImagesListLoading DocumentsListLoading MusicListLoading VideoListLoading

Estas clases son las que preparan las listas con los datos pertinentes para la actividad que las llama. Estas listas son llenadas con los datos extraídos del archivo XML ya descargado en un principio por la actividad *LauncherActicit.* Para extraer los datos del archivo XML es necesario saber los nombres de los campos a extraer del mismo y cada clase de esta tiene ya predefinidos los nombres de los campos necesarios para extraer la información que se necesita.

```
// XML node keys
// Nodo Padre
public static final String KEY_IMAGEN = "imagen";
// Nodo Hijos
static final String KEY_ID = "id";
public final static String KEY_TITLE = "title";
static final String KEY_ARTIST = "artist";
static final String KEY_PRICE = "price";
static final String KEY_THUMB_URL = "thumb_url";
public static final String KEY_LOW_IMAGE_URL = "full_imagen";
public static final String KEY_FULL_IMAGE_URL = "full_imagen_url";
public static final String KEY_IMAGES_CATEGORY = "images_category";
static final String KEY_IMAGEN_CATEGORY = "imagen_category";
static final String KEY_PRINT_PRICE = "print_price";
static final String KEY_DOWNLOAD_PRICE = "download_price";
```

Figura 4.14 Declaración de los nodos para la extracción de datos del archivo XML

Otra de las características de estas clases es que heredan la clase AsyncTask, la cual nos permite realizar actividades en otro hilo de ejecución y así de esta forma poder mostrarle al usuario alguna animación mientras esta clase realizas sus funciones. La animación que se le presenta al usuario es un *progressDialog* o dialogo de progreso. La AsyncTask nos facilita unos métodos entre los cuales se encuentran *OnPreExecute(),doInBackground() y onPostExecute().*

OnPreExecute(): se utiliza para realizar algunas tareas antes de que la clase empiece a ejecutar el código o la tarea que se desea ejecutar en otro hilo diferente al hilo principal.

doInBackground(): este método es llamada al instante de que el método anterior termina su función. Todo el código presente en este método se ejecutará en un hilo diferente al hilo principal y tiene la opción de recibir parámetros y retornarlos. Desde este método también se puede llamar al método *OnProgressUpdate()* el cual nos permite tener actualización de lo que está pasando el método *doInBackground().* Al terminar este método su función se llama al método *OnPostExecute()*.

onPostExecute(): En este método se realizan tareas finales en la clase.

```
@Override
protected void onPreExecute() {
    // TODO Auto-generated method stub
    super.onPreExecute();

    pDialog = new ProgressDialog(context);
    pDialog.setMessage(getContext().getString(R.string.cargando_imagenes_));
    pDialog.setIndeterminate(false);
    pDialog.setCancelable(false);
    pDialog.show();

}
```

Figura 4.15 Creación de un ProgressDialog en el método OnPresExecute()

Después de que termina el método *doInBackground()* de hacer su función y al pasar al método *onPostExecute()* aun estas clases guardan ciertas similitud ya que todas llaman a la clase que adapta los datos a las listas expandibles.

```
@Override
protected void onPostExecute(ArrayList<HashMap<String, String>> result) {
    super.onPostExecute(result);

    //Inicializamos el adapter y luego se le ajusta a la lista expandible
    adapter = new ImagesAdapter((Activity) exListView.getContext(),
            groupData, childData);
    exListView.setAdapter(adapter);
    pDialog.dismiss();
}
```

Figura 4.16 Creando la instancia del adapter y colocándolo a la lista expandible

4.5.2.2 ImagesAdapter DocumentsAdapter MusicAdapter VideoAdapter

Al tener listas desplegables o de otro tipo es necesario utilizar un adaptador para poder rellanarlas con el contenido que deseamos. En nuestro utilizamos cuatros lista con ciertas variaciones en cada una por eso la necesidad de utilizar un adaptador especifico para cada una. Es ciertas variaciones vienen debido a que en cada una se espera mostrar cierta información diferente tanto en contenido como en posición.

También cabe destacar que las listas expandible se forman por grupo e hijos y tanto para el grupo como para el hijo se ha generado una vista personalizada la cual es diferente para cada caso.

4.5.3 *ImagesFullActivity*

Esta actividad nace con la intención de mostrarle al usuario las imágenes de una categoría de manera cómoda y convencional. Esta actividad es llamada al momento que se pulsa sobre una imagen de la lista desplegable en la interfaz que pertenece a la actividad *ImagesActivity()*. En esta activa se hace uso de lo que es una galería para mostrar todas las imágenes y de un *imagenSwicher* para ir colocado las imágenes mientras pasamos de una a otra.

Al igual que las listas expandibles las galerías necesitan de un adaptador y debido a eso existe la clase *ImagesGalleryAdapter()* la cual tiene la función de ser el adaptador de esta galería. Por otra parte en esta actividad se emplea una *AsyncTask* debido a que la imagen que se va a presentar en el ImageSwicher debe ser descarga si no se encuentra en cache y con la ayuda de la *AsyncTask* podemos descargarla en otro hilo y presentarle un *ProgressDialog* mientras esta se descarga.

4.5.4 *MusicPlayActivity*

Utilizando el objeto MediaPlayer esta actividad es capaz de reproducir audio desde una URL. La configuración utilizada permite que se pueda reproducir el archivo de audio desde que se tiene un buffer aceptable evitando así tener que descargar el archivo de audio por completo y Después reproducirlo.

4.5.4 *PayActivity*

Esta actividad es la que emplea la interacción con la librería de PayPal ofreciendo así la posibilidad de poder realizar cobros.

Al momento de implementar PayPal se utilizaron unas cuentas “falsa” ofrecidadas por *SandBox* que no es más que un entorno de prueba de PayPal donde se simula a igual los servicios que nos ofrece PayPal.

En la figura 4.17 se puede apreciar un trozo del código de esta actividad donde se le pasan los parámetros a la librería de PayPal para la gestión del cobro. Se puede notar que la cuenta a la que se le acreditará el pago es “enr.ri_1341578969_biz@gmail.com” la cual fue facilitada por *SandBox*.

```
public void PayPalButtonClick(View arg0) {
    // Create a basic PayPalPayment.
    PayPalPayment payment = new PayPalPayment();
    // Sets the currency type for this payment.
    payment.setCurrencyType("EUR");
    // Sets the recipient for the payment. This can also be a phone
    // number.
    payment.setRecipient("enr.ri_1341578969_biz@gmail.com");
    // Sets the amount of the payment, not including tax and shipping
    // amounts.
    BigDecimal st = new BigDecimal(Double.parseDouble(itemFinalPrice));
    st = st.setScale(2, RoundingMode.HALF_UP);
    payment.setSubtotal(st);
    // Sets the payment type. This can be PAYMENT_TYPE_GOODS,
    // PAYMENT_TYPE_SERVICE, PAYMENT_TYPE_PERSONAL, or
    // PAYMENT_TYPE_NONE.
    payment.setPaymentType(PayPal.PAYMENT_TYPE_GOODS);
    // Sets the merchant name. This is the name of your Application or
    // Company.
    payment.setMerchantName("Oliva");
    // Sets the description of the payment.
    payment.setDescription(itemType + " - " + itemName);
```

Figura 4.17 Fragmento PayActivity – Parámetros par el cobro

4.6 Implementación de la interfaz de usuario

A continuación se explica las diferentes interfaces que componen la interfaz de usuario de la aplicación, relacionando cada interfaz con la actividad que la utiliza. En la figura 4.30 se muestra la relación entre las interfaces.

4.6.1 Interfaz Principal

Como ya hemos comentado en un apartado anterior mente para la interfaz principal se quiso emplear un *TabHost* con el fin de que el usuario no tuviese que usar un menú para acceder al contenido que desea sino que tan solo viera los diferentes tabs y al presionar sobre ellos apareciera una actividad nueva sin la necesidad de cambiar la vista principal.

En la figura 4.18 se puede observar esta implementación, lo que está encerrado en rojo no pertenece al Layout de esta interfaz sino que es el layout de la interfaz de inicio.

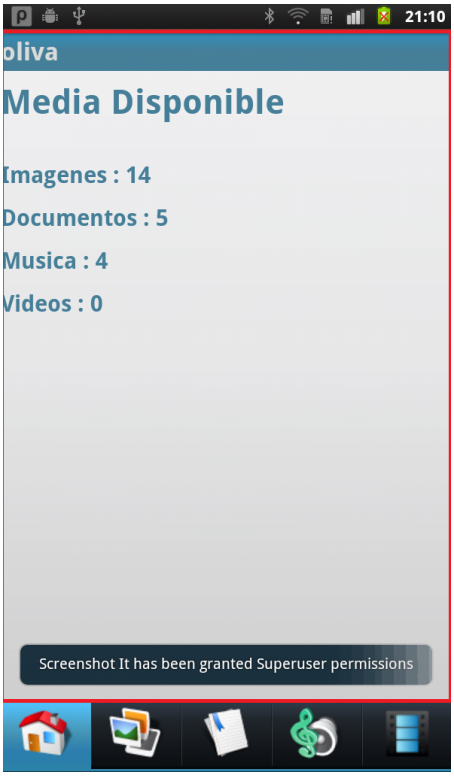

Figura 4.18 interfaz Principal

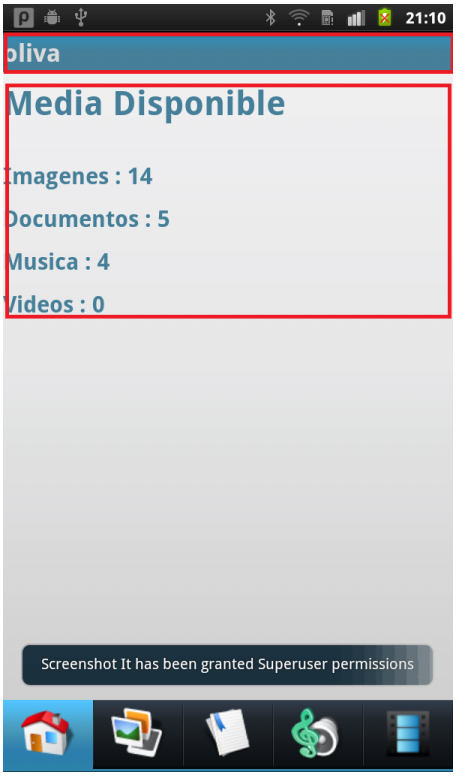

Figura 4.19 interfaz Inicio

4.6.2 Interfaz Inicio

En esta interfaz, figura 4.19 partes encerradas en rojo, la intención es mostrarle un resumen al usuario de la cantidad de archivos disponibles por categorías. El layout que implementa esta interfaz es home_activity y está compuesto de un *RelativeLayout* el cual contiene otro RelaticeLayout en el cual están un grupo de *TextView* que son los textos que se pueden apreciar en la imagen.

4.6.3 Interfaz Imagen, Música, Documentos, Video

Todas estas interfaces implementas listas Expandibles por lo tanto los layout de sus interfaces son muy similares, por esta razón solo se explicará el layout de la interfaz imagen. En la figura 4.20 podemos apreciar como ha quedado esta interfaz. La lista desplegable está compuesta por un layout para los grupos, en este caso el images_group_layout.xml y un layout para los hijos images_child_layout.xml, el layout que contiene la lista expandible se llama images_activity.xml.

Los grupos tan solo cuentas con *ImagenView* para mostrar un pequeño icono y un TextView para mostrar el nombre del grupo y los hijos cuentan con un *ImagenView* para mostrar un Thumbnail de la imagen a mostrar, para el caso de los documentos videos y música también muestra un Thumbnail, y dos *ImagenView* más para mostrar unos pequeños iconos donde se hace referencia al precio de descarga e impresión. También cuenta con tres *TextView* para mostrar el nombre del archivo y los precios de descarga e impresión.

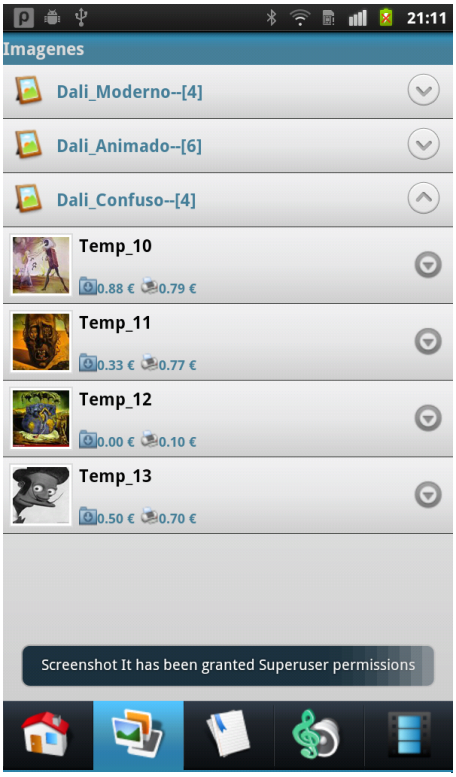

Figura 4.20 Interfaz Imágenes

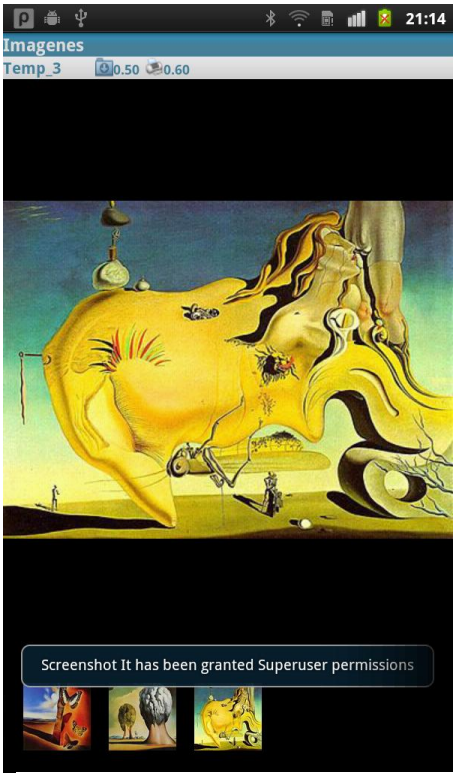

Figura 4.21 Interfaz Imagen Completa

4.6.4 Interfaz Imagen Completa

Pensando en que al usuaria le gustaría poder ver las imágenes en una galería y la vez que se le mostrase una imagen un poco más grande mientras navega por la galería es como surge esta interfaz la cual se basa en el layout images_full_activity.xml y que implementa la actividad *FullImageActivity*.

En esta vista, figura 4.20 se puede apreciar el uso una *ImagenSwitcher* para mostrar la imagen agrandada y la parte inferior la galería. En la parte superior se puede apreciar dos iconos los cuales representan la descarga y la impresión acompañados de dos TextView que representan los diferentes precios.

4.6.5 Interfaz Tocar Audio

Esta interfaz, figura 4.20 ha sido diseñada con el fin de crear un reproductor de audio propio para la aplicación, personalizado de tal forma que presentara el nombre del artista, la canción que está reproduciendo una barra de progreso y los típicos botones de un reproductor de audio. La actividad que implementa esta vista es *MusicPlayActivity* y layout de esta interfaz es el music_play_activity.xml

Figura 4.22 Interfaz Tocar Audio

4.6.6 Interfaz de Pago

A diferencia de las otras actividades que solo utilizan una interfaz la actividad *pay_activity* utiliza múltiple interfaz para poder cumplir con sus funciones. La figura 4.21 muestra la interfaz inicial con la inicia esta actividad en la que hay dos botones, con los iconos de descargar o imprimir y su respectivos precios.

Figura 4.23 Interfaz Pago-Elegir Servicio

Al momento de elegir unos de las dos acciones que presentan estos botones puede que surjan dos cosas, que sea posible realizar el pago o que el servicio no esté disponible. La Figura 4.22 muestra la interfaz que aparecería en caso de que no esté disponible la descarga y la Figura 4.23 en el caso que no esté disponible la impresión.

Figura 4.24 Interfaz Pago-Descarga no disponible

Figura 4.25 Interfaz Pago-Impresión no disponible

La figura 4.24 muestra la interfaz que aparecerá Después de que se realice un pago para imprimir un archivo.

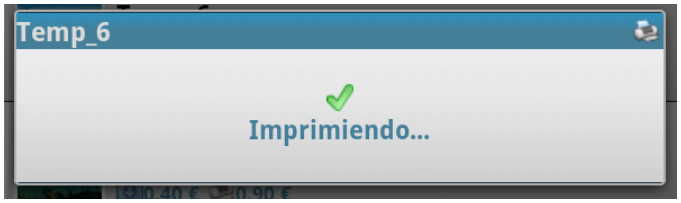

Figura 4.26 Interfaz Pago-Imprimiendo

Volviendo a la Figura 4.21, en caso de que el servicio esté disponible la actividad mostrará la interfaz que se muestra en la Figura 4.25. Desde esta interfaz al presionar el botón de PayPal se carga una interfaz propietaria de PayPal que se muestra en la figura 4.26.

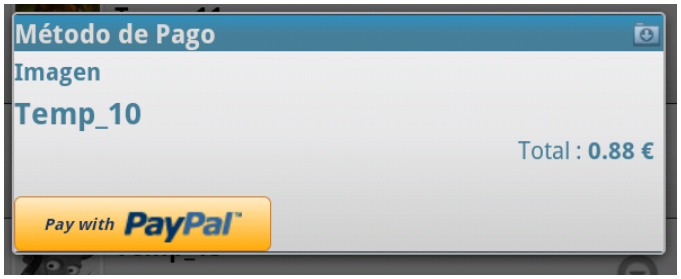

Figura 4.27 Interfaz Pago-PayPal

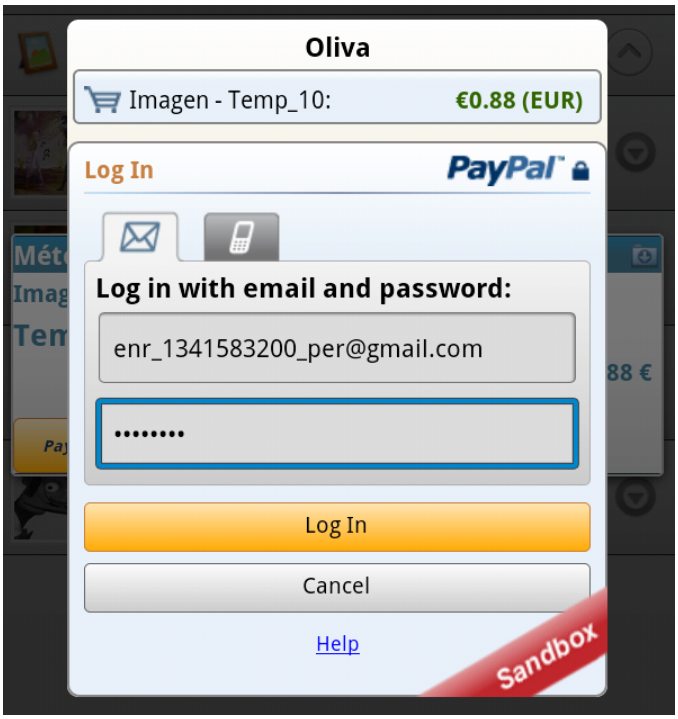

Figura 4.28 Interfaz Pago-PayPal-2

4.7 Manifiesto de la aplicación

Como ya se ha comentado este archivo es de suma importancia dentro de la aplicación, debido a que en este se declaran los componentes de Android utilizados como los recursos y los permisos que la aplicación necesita. En nuestro caso se necesitaron unos 5 permisos para el buen funcionamiento de la aplicación.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tfm.oliva" android:versionCode="1" android:versionName="1.0">

    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15" />

    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Figura 4.29 Fragmento del Manifiest, permisos de la aplicación

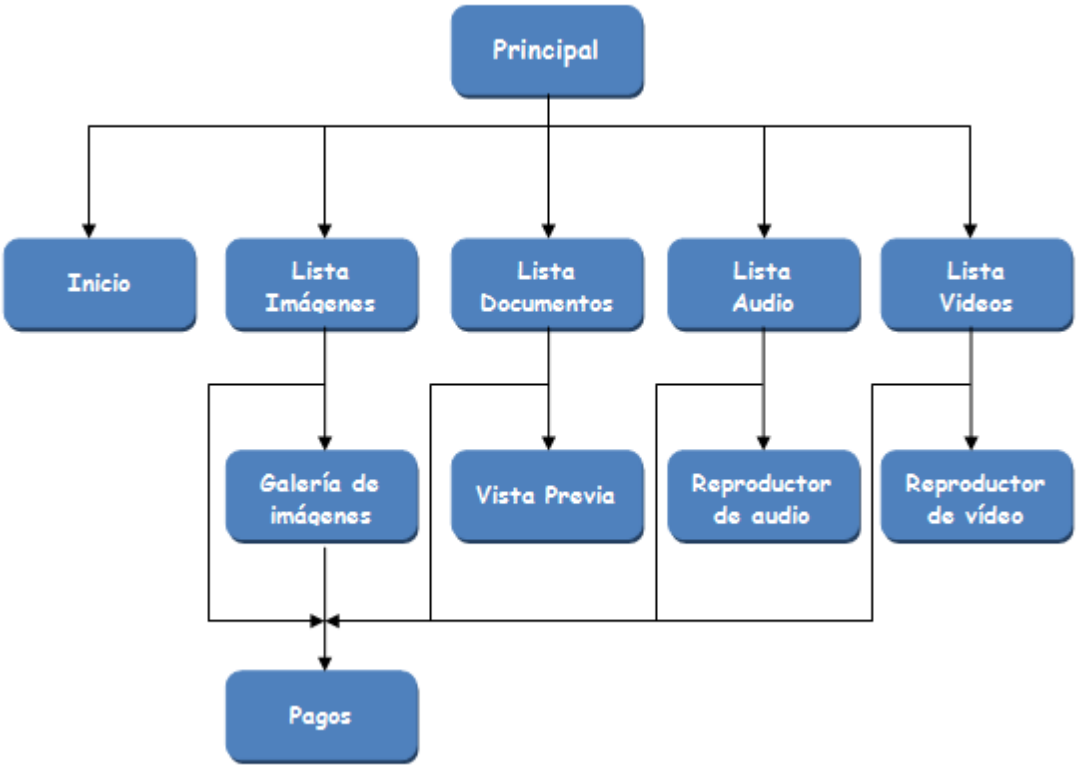

Figura 4.30 Relación entre las interfaces de la aplicación

Capítulo 5.

Prototipo de la Aplicación

Después de haber explicado la implementación de la aplicación procederemos a presentar el prototipo funcional de esta. La forma en que se mostrará su funcionalidad será a través de un ejemplo guiado.

5.1 Adquiriendo una Imagen desde la aplicación

En este ejemplo veremos cuáles son los pasos a dar para adquirir una de las imágenes ofertadas en la aplicación y el proceso de pago que esto representa.

Después de tener la aplicación instalada en un terminal Android, para este caso una Tables de 7 pulgadas.

1- Procedemos a inicializar la aplicación, Figura 5.1.

2- Esperamos mientras la aplicación se conecta con el servidor y descarga el archivo XML, Figura 5.2.

3- Elegimos el *tab* de las imágenes, el cual a simple vista por el icono que lo representa es fácilmente deducible cual es. Figura 5.3

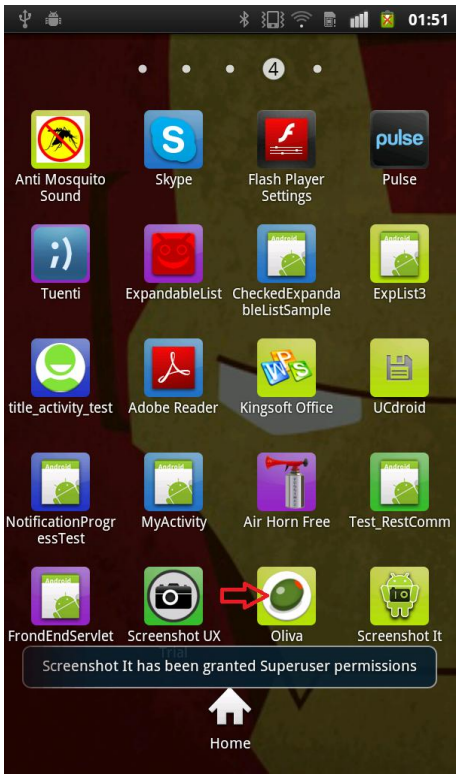

Figura 5.1 Iniciando la aplicación

Figura 5.2 Imagen de inicio de la Aplicación

4- Después de seleccionar el tab de imágenes procedemos a seleccionar la imagen que nos interesa de los grupos o categorías disponibles.

5- Antes de seleccionar la imagen tenemos dos posibles opciones, o bien pulsamos sobre el botón de menú a la derecha, Figura 5.4 o pulsamos sobre el ítem directamente Figura 5.5.

6- Al pulsar sobre el botón menú, Figura 5.4, nos presenta la interfaz de para seleccionar la opción de si queremos descarga la imagen o imprimirla, en esta interfaz también tenemos el precio de cada opción. Figura 5.6

7- Después del paso seis tenemos un resumen de la opción seleccionada y nos aparece el botón de PayPal para realizar el Pago. Figura 5.7

8- Al pulsar el botón PayPal toma el control y aparece una interfaz propia de ellos donde procedemos a introducir nuestras credenciales y se procede a realizar el pago.

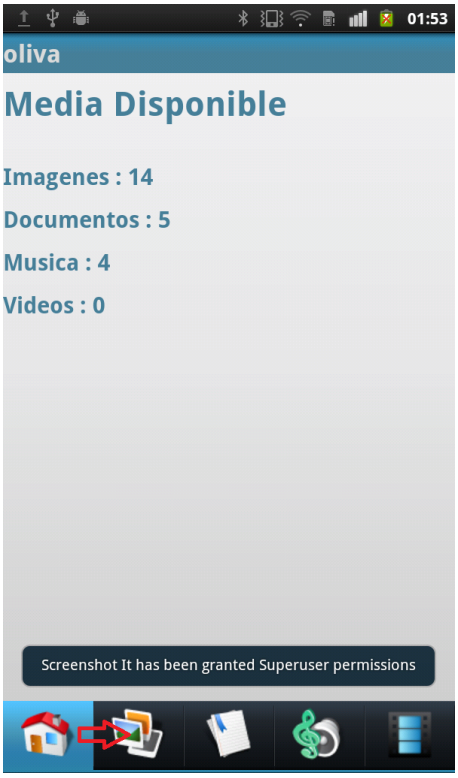

Figura 5.3 Selección del tab Imágenes

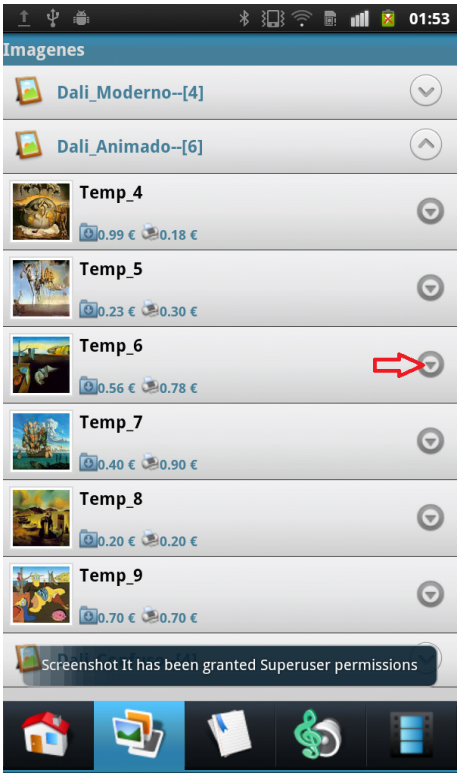

Figura 5.4 Selección de la imagen, botón menú

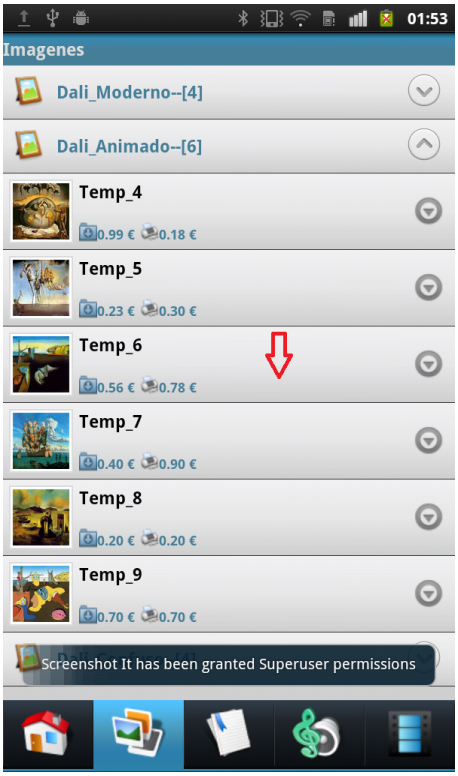

Figura 5.5 Selección de la imagen, ítem

Figura 5.6 Selección de la opción de compra sobre el ítem

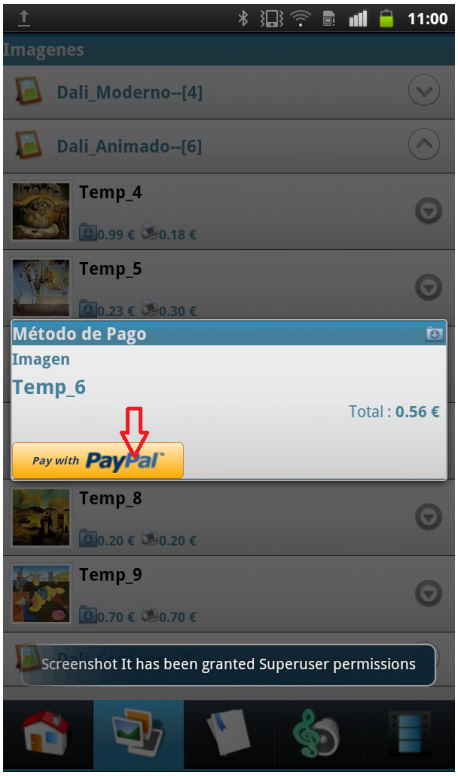

Figura 5.7 Resumen del opciones seleccionada

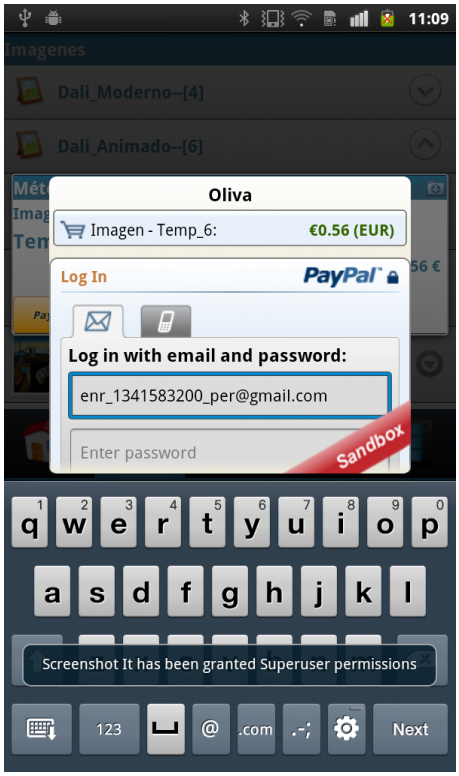

Figura 5.8 Interfaz de PayPal para realizar el Pago

9- Después de introducir las credenciales tocas empieza el proceso de verificación, Figura 5.9 y termina con la confirmación del pago. Figura 5.10.

10- Al confirmar el pago y este ser satisfactorio empieza el proceso de descarga. Figura 5.11

11- Después de la imagen descargarse está se mostrará al usuario.

12- Retomando el punto cinco, en vez de seleccionar el botón menú ahora seleccionamos el ítem como muestra la figura 5.5. Esto nos llevara a una galería donde podremos navegar por las imágenes disponibles en la categoría a la que pertenece la imagen seleccionada. Figura 5.12 y Figura 5.13

13- Al momento de pulsar sobre una imagen se vuelve a repetir el proceso desde el punto seis.

En cuanto a querer obtener cualquiera archivo de las otras categorías el proceso es bastante similar.

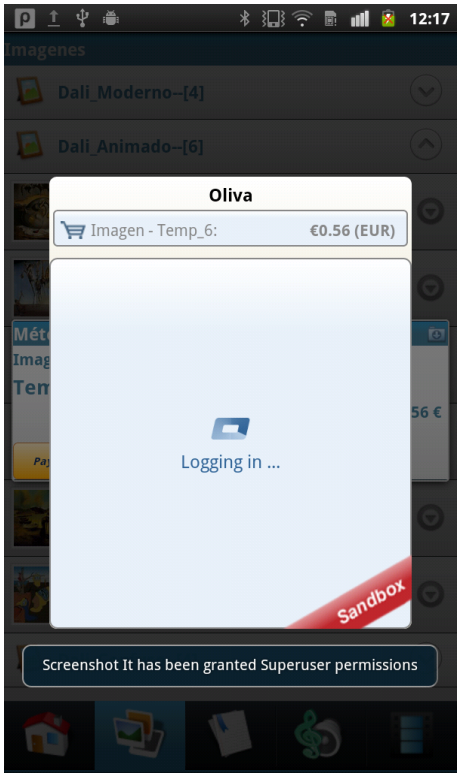

Figura 5.9 Verificación de las credenciales por parte de PayPal

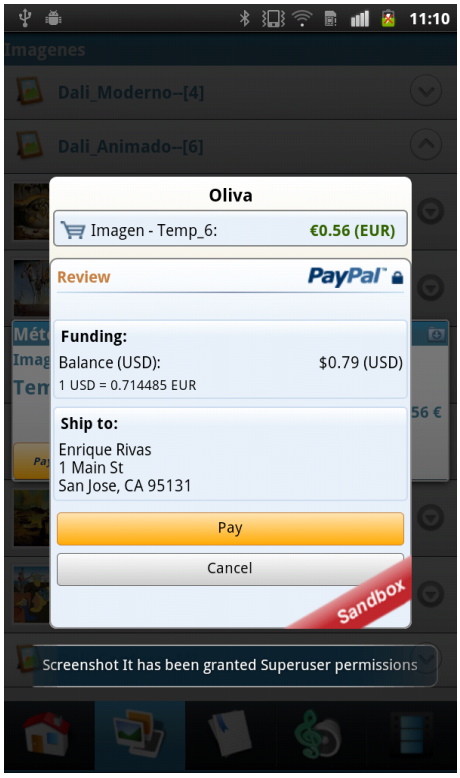

Figura 5.10 Confirmación del pago

Figura 5.11 Proceso de descarga

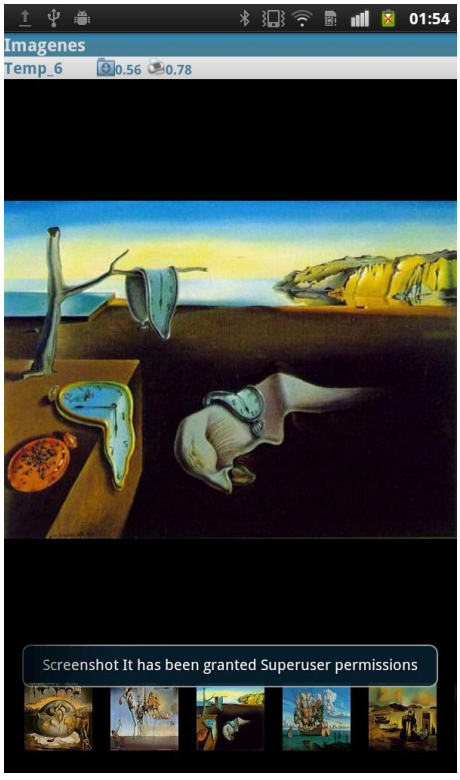

Figura 5.12 Galería de Imágenes

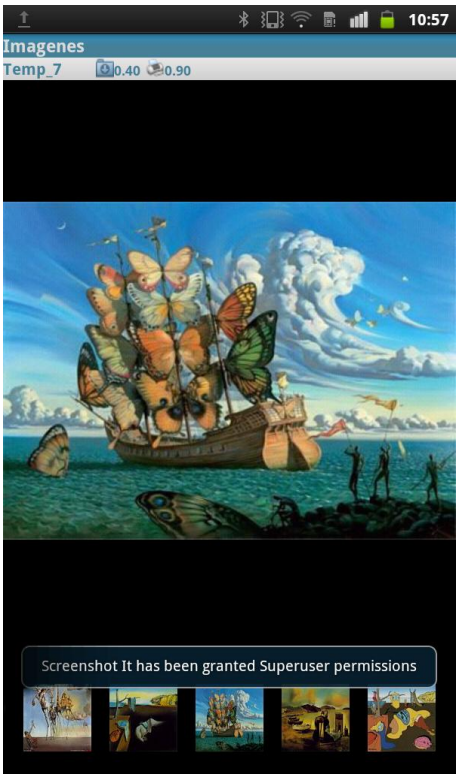

Figura 5.13 Navegando por la galería

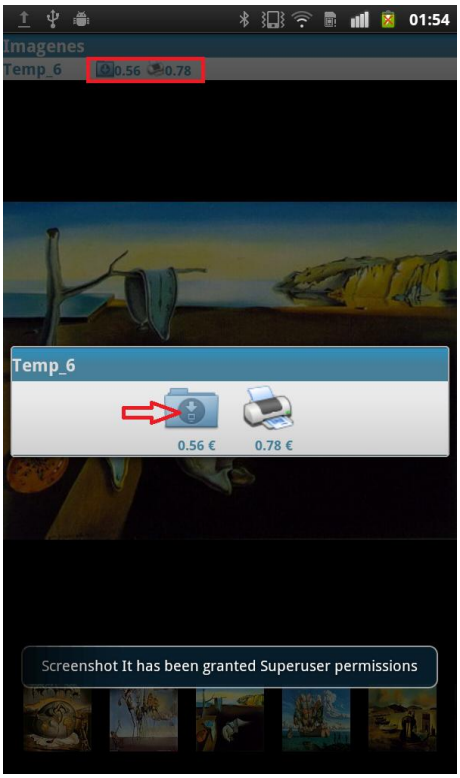

Figura 5.14 Selección de la opción de compra sobre el ítem

Capítulo 6.

Conclusiones

Gracias a la gran comunidad de desarrolladores de aplicaciones para Android y la buena documentación que ofrece la comunidad oficial de Android, desarrollar este proyecto con conocimiento prácticamente nulos sobre el tema pudo ser realidad. La idea por la que se eligió esté proyecto fue por la motivación de aprender a desarrollar aplicaciones para Android, lo que ha sido posible, también agregando un extra que ha venido indirectamente, aprender el lenguaje de programación Java y nociones sobre XML.

Debido a los pocos conocimientos sobre Java en muchas ocasiones fue difícil interpretar totalmente algún ejemplo o poner en funcionamiento algún trozo de código, lo que le agregó horas significativas al desarrollo final de la aplicación, por lo que es una buena idea tener una buena base sobre el lenguaje de programación Java.

Los objetivos iniciales fueron alcanzados, aunque a simple vista se pude apreciar que la aplicación a sido "sencilla" pero tomando en cuenta que no se tenía conocimientos previos sobre Android ni Java y tampoco sobre implementación de servidores y se logró crear una aplicación que es capaz de implementar de forma sencilla para el usuario gestión de archivos multimedia desde un servidor, gestión de pagos a través de *PayPal*, una interfaz grafica cómoda para el usuario implementación de funcionalidades complejas como lo son las *ListasExpandibles* personalizadas, los *ImagenSwitcher*, notificaciones, creación de un reproductor propio de audio entre otras.

En cuantos mis objetivos personales me encuentro muy satisfecho con los conocimientos adquiridos y mucho mejor ya que mi motivación por emprender una carrera como desarrollado de aplicaciones para Android está mucho mejor después de la finalización de este proyecto.

6.1 Mejoras aplicables

6.1.1 Utilización de imágenes propias

Unas de las limitaciones que presenta la aplicación es que solo se pueden imprimir las imágenes alojadas en el servidor. Por lo que agregar la funcionalidad de que el usuario pueda imprimir sus propias imágenes daría paso a una mejor experiencia por parte del usuario.

6.1.2 Presentación de las excepciones al usuario

Por el momento, solo se manejaron las excepciones cuando no tenemos conexión con el servidor pero de al usuario se le facilita poca información cuando la aplicación tiene una excepción. Una buena presentación nos facilitaría hacerle mejoras fácilmente a la aplicación ya que los mismos usuarios podrían comunicar específicamente que ha sucedido.

6.1.3 Nuevas formas de pagos

Debido a la gran aceptación que tiene PayPal fue la forma de pago elegida como primera fuente de pago para la aplicación. Pero no todos utilizan este método para realizar sus compras virtuales. Otros de los métodos que se podrían usar son GoogleWallet o pagos por SMS.

6.1.4 Historial de Compras

Después de realizar sus compras el usuario no tiene forma de cómo saber que ha imprimido o descargado, de forma que sería de utilidad para el usuario poder contar con un breve historial de sus acciones.

Bibliografía

[1] Android Application Development for For Dummies
Wiley Publishing, Inc, 111 River Street Hoboken, NJ 07030-5774

[2] Programming Android: Zigurd Mednieks
Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472

[3] Java™ y DOM : En este tutorial se explica qué es DOM y cómo usarlo desde Java™.
Revisión 1.0 28-09-2003 Revisado por: Alberto Gimeno

[4] Wikipedia
Android
<http://es.wikipedia.org/wiki/Android>[Online] [Consulta: 19-07-12]

[5] Brighthub
Google Android's Internal Structure in Plain English
<http://www.brighthub.com/mobile/google-android/articles/17822.aspx>
[Online] [Consulta: 19-07-12]

[6] Android developers
Application Fundamentals
< http://developer.android.com/guide/topics/fundamentals/activities.html>[Online]
[Consulta: 19-07-12]

[7] El economista
Cuatro de cada cinco smartphones vendidos en España son Android
< http://www.eleconomista.es/CanalPDA/2012/38725/cuatro-de-cada-cinco-smartphones-vendidos-en-espana-son-android/ >[Online] [Consulta: 19-07-12]

[8] Android developers
<http://developer.android.com/guide/topics/fundamentals/services.html>
[Online] [Consulta: 19-07-12]

[9] Sgoliver
Componentes de una aplicación Android
<http://www.sgoliver.net/blog/?p=1295> [Online] [Consulta: 19-07-12]

[10] Android developers <http://developer.android.com/guide/topics/providers/content-providers.html> [Online] [Consulta: 19-07-12]

[11] Android developers
<http://developer.android.com/guide/topics/manifest/manifest-intro.html>
[Online] [Consulta: 19-07-12]

[12] Android developers
<http://developer.android.com/guide/topics/fundamentals/activities.html>
[Online] [Consulta: 19-07-12]

[13] Android developers
< http://developer.android.com/reference/android/view/View.html >
[Online] [Consulta: 19-07-12]

[14] Bogotobogo
User Interface
<http://www.bogotobogo.com/Android/android4UserInterface.php>
[Online] [Consulta: 19-07-12]

[15] Cyrilmottier
ListView Tips & Tricks #4: Add several clickable areas
<http://android.cyrilmottier.com/?p=525>[Online] [Consulta: 19-07-12]

[16] Hrupin
Example of streaming mp3 mediafile from URL with Android MediaPlayer class
<http://www.hrupin.com/2011/02/example-of-streaming-mp3-mediafile-with-android-mediaplayer-class >[Online] [Consulta: 19-07-12]

[17] Joshclemm
Custom Android Tabs
< http://joshclemm.com/blog/?p=136>[Online] [Consulta: 19-07-12]

[18] Tutomobile
Réaliser une custom ProgressBar/SeekBar sur Android
<http://www.tutomobile.fr/realiser-une-progressbar-seekbar-personnalisee-sur-android-tutoriel-android-n%C2%B024/10/02/2011/>[Online] [Consult: 19-07-12]

[19] Paullabis
Tutorial on Android Views - Part 2
<http://www.paullabis.com/2010/04/tutorial-on-android-views-part-2.html>[Online]
[Consulta: 19-07-12]

[20] Powenko
Media, play http video
<http://www.powenko.com/en/?p=3249>[Online] [Consulta: 19-07-12]

[21] Stackoverflow
Mobile Express Checkout Library
<http://stackoverflow.com/questions/9737052/android-mobile-express-checkout-library>[Online] [Consulta: 19-07-12]

[22] Francho
Ejemplo de uso de la libreria de pagos de Paypal para Android
< http://francho.org/2010/12/29/ejemplo-de-uso-de-la-libreria-de-pagos-de-paypal-para-android/ >[Online] [Consulta: 19-07-12]

[23] Stackoverflow
Java check if file exists on remote server using its url
< http://stackoverflow.com/questions/4596447/java-check-if-file-exists-on-remote-server-using-its-url>[Online] [Consulta: 19-07-12]

[24] Blog.csdn
ExpandableListActivity
<http://blog.csdn.net/zzzl/article/details/6321112>[Online] [Consulta: 19-07-12]

[25] Huuah
Android Progress Bar and Thread updating
<http://huuah.com/android-progress-bar-and-thread-updating/>[Online] [Consulta: 19-07-12]

[26] Chuwiki
Ficheros XML
< http://chuwiki.chuidiang.org/index.php?title=Ficheros_XML>[Online] [Consulta: 19-07-12]

[27] Developer.Android
User Interface
< http://developer.android.com/guide/topics/ui/declaring-layout.html>[Online] [Consulta: 19-07-12]

[28] Developer.Android
Dowload-Get the Android SDK
< http://developer.android.com/sdk/index.html>[Online] [Consulta: 19-07-12]

[29] Reecon
Uploading files to HTTP server using POST. Android SDK.
< http://reecon.wordpress.com/2010/04/25/uploading-files-to-http-server-using-post-android-sdk/ >[Online] [Consulta: 19-07-12]

[30] Stackoverflow
Uploading image to apache tomcat server with android
< http://stackoverflow.com/questions/8393112/uploading-image-to-apache-tomcat-server-with-android >[Online] [Consulta: 19-07-12]

[31] Androidhive
Android Custom ListView with Image and Text
<http://www.androidhive.info/2012/02/android-custom-listview-with-image-and-text/>[Online]
[Consulta: 19-07-12]

[32] Android-adda
Custom Expandable Listview
<http://android-adda.blogspot.com.es/2011/06/custom-expandable-listview.html>[Online]
[Consulta: 19-07-12]

[35] Vogella
Android Intents – Tutorial
<http://www.vogella.com/articles/AndroidIntent/article.html>[Online] [Consulta: 19-07-12]

[36] Android-coding
Overwrite MENU key to create custom menu
<http://android-coding.blogspot.in/2011/07/overwrite-menu-key-to-create-custom.html>[Online]
[Consult: 19-07-12]

[37] Edu4java
Java Servlet
<http://edu4java.com/servlet.html> [Online] [Consulta: 19-07-12]

[38] fr4gus
Usando efectivamente el asynctask parte 1
<http://www.fr4gus.com/2011/05/13/usando-efectivamente-el-asynctask-parte-1/>
[Online] [Consulta: 19-07-12]