Trabajo Final de Carrera

Desarrollo de Aplicaciones para dispositivos móviles

Autor: Daniel Borraz Morón
Consultor: Albert Grau Perisé

A ti, Sonia, mi niña, por tu cariño, apoyo, paciencia y comprensión.

A ti, mi hija Nerea, porque no quiero pasar un día más sin dedicarte todo mi tiempo.

A vosotros, mis padres y hermanos, siempre habéis estado ahí.

A ti, mi hermano, desde donde estés, me has ayudado a ser mejor persona.

Sois mi fuerza, os lo debo todo.

Índice de contenidos

1. Definición general del proyecto

a. Introducción, motivación, contexto y objetivos

Las aplicaciones móviles se han convertido hoy en día en algo de lo más cotidiano, éstas nos facilitan la comunicación social y además nos han acercado todo tipo de herramientas que antes solo estaban en un ordenador personal, y se han creado otras específicas que apoyándose en las propiedades de movilidad que aporta el terminal no se podrían llevar a cabo en otro tipo de dispositivo.

La motivación que me lleva a haber elegido como área de *"Desarrollo de aplicaciones para dispositivos móviles"* de Trabajo Final de Carrera es debido a que como he comentado en la introducción las Apps móviles son ya parte de nuestra vida, han crecido y evolucionado enormemente en un periodo de tiempo muy corto, y poseen un porvenir de lo más prometedor, con lo cual, como futuro "Ingeniero en Informática de Sistemas" me interesa y tengo como objetivo el llevar a cabo el desarrollo de una aplicación de este tipo con éxito, y añadir este tipo de tecnología para aportar y mejorar mi experiencia profesional.

He elegido la plataforma Android, básicamente porque es con aquella que me encuentro más familiarizada, los dispositivos que poseo son todos de este tipo, permite el desarrollo de forma gratuita, además se ha convertido en una plataforma que más ha crecido últimamente, tiene infinidad de recursos ya que se trata de una plataforma *Open Source* basada en Linux.

Por todo ello, mi planteamiento inicial es realizar una aplicación que aunque parece inicialmente sencilla puede resultar de gran utilidad y muy potente, que nos será de gran utilidad y realizará tareas que a veces por falta de dedicación pueden ser desastrosas.

Copy2Cloud**,** que es como he denominado en un principio mi aplicación, básicamente realizará varias copias de elementos indispensables de nuestro dispositivo móvil en un servicio de ficheros tipo nube. Como bien sabemos el espacio físico de los dispositivos es cada vez más limitado, las aplicaciones y datos cada vez ocupan más, y esto nos lleva a tener descontrol sobre el almacenaje y veracidad de éstos, es decir, tenemos tal cantidad de fotos y videos personales en nuestros terminales que en caso de desastre la mayoría de las personas sin conocimientos técnicos darían por perdida toda esa acumulación de nuestros recuerdos, de parte de nuestras vidas.

b. Referencias

En Google Play, hay aplicaciones parecidas, busco realizar una aplicación mucho más sencilla, que aunque en principio funcionará sobre la API de Google Drive, me gustaría realizarlo en otros servicios de nube gratuitos como Dropbox, etc.

Otras aplicaciones de éste tipo que he encontrado en la *"Store de Google"* son:

- **G Cloud Backup.**
  https://play.google.com/store/apps/details?id=com.genie9.gcloudbackup
- **Storino**.
  https://play.google.com/store/apps/details?id=pl.storino.android&hl=es
- **Zip Cloud.**
  https://play.google.com/store/apps/details?id=com.jdibackup.android.zipcloud&hl=es

De todas estas para mí la más completa es G Cloud Backup.

Características técnicas

De Software

- Para el desarrollo de la aplicación utilizaremos IDE de Eclipse con el plugin ADT (Android Developer Tools) para agilizar el desarrollo de aplicaciones Android. (SDK incluida).
- API de Google Drive.

De Hardware

- Para el desarrollo de la aplicación utilizaremos un PC con procesador AMD de 64 bits con Windows 7 Professional de 64 bits.
- Para la realización de las pruebas se dispone de dos dispositivos móviles con sistema Android Gingerbread 2.3.5 modelos Huawei Ideos X5 y HTC Desire HD.

2. Funcionalidad

a. Características funcionales y requerimientos.

Algunas de las funciones que pretendo implementar en Copy2Cloud:

- **Camera2Cloud:** Fotos y videos que directamente se subirán al servicio de nube.
- **Play2Cloud:** Reproducir música de ficheros mp3 almacenados en nuestra nube.
- **Contacts2Cloud:** Copias de seguridad de nuestros contactos y servicios de mensajería SMS.
- **Files2Cloud:** Copias de seguridad de nuestras carpetas de ficheros.
- **Write2Cloud:** Generar, abrir y editar documentos “on-line”.
- **Sync2Cloud:** Sincronizar y programar las copias.
- **Look2Cloud:** Navegar por nuestro sistema de ficheros en la nube.
- **Tools2Config:** Configuraciones personales de la aplicación.

Los requerimientos de *Copy2Cloud* son:

- Un terminal Android con versión 2.2 (Froyo) con pantalla táctil.
- El espacio necesario no superará los 4 MBytes de memoria interna.
- Cámara integrada (no necesario para la mayoría de las funciones).
- La aplicación requerirá permisos de acceso a la cámara y a la nube.

b. Diseño e implementación de la interfaz gráfica de la aplicación.

El diseño e implementación de la interfaz gráfica de *Copy2Cloud* se pretende que sea lo más sencilla e intuitiva posible, con unas simples pulsaciones realice una acción determinada, para que el usuario final no necesite manual alguno aunque esté disponible, y que con la simple acción de tocar alcance aquello que busca realizar. A continuación se explica brevemente el manejo de la interfaz gráfica y la navegabilidad.

La interfaz de *Copy2Cloud* se presentará tal como se aprecia en las capturas, iconos amplios y *"tocables"*, que facilitarán la tarea en dispositivos de menor pantalla o a usuarios con problemas de visibilidad, con colorido e iconografía sencilla, que pese a titular cada una de las acciones deja claro la función de cada botón.

La navegabilidad entre pantallas, como todo lo que se pretende en la aplicación, también es muy sencilla, con un simple gesto con nuestros dedos de izquierda a derecha se accede a un panel u otro.

Cada aplicación llevará a un interfaz sencillo de exploración de ficheros o a funciones propias del móvil sin que tengan mayor importancia de implementación gráfica en el diseño, por ejemplo, Camera2Cloud abre directamente la cámara del dispositivo móvil y lo guarda directamente en la nube, si da error o lo realiza con éxito preguntará si queremos hacer más fotos o videos y de lo contrario volverá al menú principal.

Si se lograra implementar todas las funciones exitosamente, se deja abierta la posibilidad de agregar más o de mejorar las propuestas.

3. Planificación y temporización

a. Resumen del proyecto

Trabajo Final de Carrera - Fechas de entregas	
Fecha inicio	28/02/2013
Fecha entrega Plan de trabajo TFC (PAC 1)	11/04/2013
Fecha entrega Diseño centrado en el usuario (PAC 2)	08/04/2013
Fecha entrega Desarrollo de la Aplicación (PAC 3)	20/05/2013
Entrega Memoria y Presentación TFC (ENTREGA FINAL)	10/06/2013
Debate virtual y finalización TFC	27/07/2013

b. Listado de tareas

Las tareas a realizar en función de las principales actividades que compondrán el proyecto son las siguientes:

- **Planificación TFC (PAC 1)**

Siguiendo la metodología de planificación aprendida, se determinarán que tareas son necesarias para alcanzar los objetivos del proyecto y se estimará el esfuerzo y los recursos necesarios para llevarlas a cabo.

- **Metodología del diseño centrado en el usuario en la aplicación definida en el Plan de Trabajo (PAC 2)**

El objetivo principal de esta actividad es: saber aplicar el Diseño Centrado en el Usuario (DCU) en el análisis, diseño, desarrollo y evaluación de sistemas móviles. Para el desarrollo de esta tarea se dispondrá de dos módulos didácticos:

- Diseño centrado en el usuario.
- Diseño centrado en el usuario para dispositivos móviles.

Este objetivo se concreta en los siguientes apartados:

1) Investigar usuarios y recoger requisitos, tanto cuantitativos como cualitativos, que ayudarán a conocer los usuarios y definir perfiles.

2) Examinar y analizar las condiciones en que se utilizará el sistema para definir su contexto de uso.

3) Elaborar un análisis de tareas.

4) Elaborar escenarios de uso.

5) Definición de los flujos de interacción en el sistema.

6) Diseñar y construir un prototipo de alto nivel del sistema teniendo en cuenta los conceptos de las evaluaciones heurísticas y de las particularidades del diseño para dispositivos móviles.

7) Plantear la evaluación del prototipo del sistema mediante un test con usuarios.

8) Mantener la visión de conjunto en todas las etapas de la elaboración de la práctica, identificando los aspectos a mejorar en cada iteración del proceso de DCU.

Esta PAC está formada por cuatro partes que siguen las fases del DCU:

Análisis ➔ diseño ➔ evaluación:

1. Usuarios y contexto de uso (Análisis).

2. Diseño conceptual (Diseño).

3. Prototipo (Diseño).

4. Evaluación (Evaluación).

- **Implementación de la aplicación (PEC 3)**

Esta tarea comprende las siguientes actividades:

1) Una revisión de los requerimientos proporcionados en el enunciado, para evaluar si son suficientes para comenzar el diseño e identificar los puntos abiertos que requieran una clarificación.

2) La construcción de las interfaces de la aplicación.

3) La realización de las pruebas unitarias de las clases construidas.

4) La realización de las pruebas integradas de funcionamiento de la aplicación una vez probadas las clases individuales.

5) Una vez finalizado el desarrollo, se realizará una revisión del código desarrollado y de la documentación para verificar que son completos, homogéneos y consistentes.

6) Para finalizar, se realizará la entrega prevista por la evaluación continua

- **Elaboración Memoria y Presentación TFC (ENTREGA FINAL)**

Esta tarea final recoge la realización y entrega de los documentos finales de este Trabajo Final de Carrera.

c. Planificación temporal del proyecto

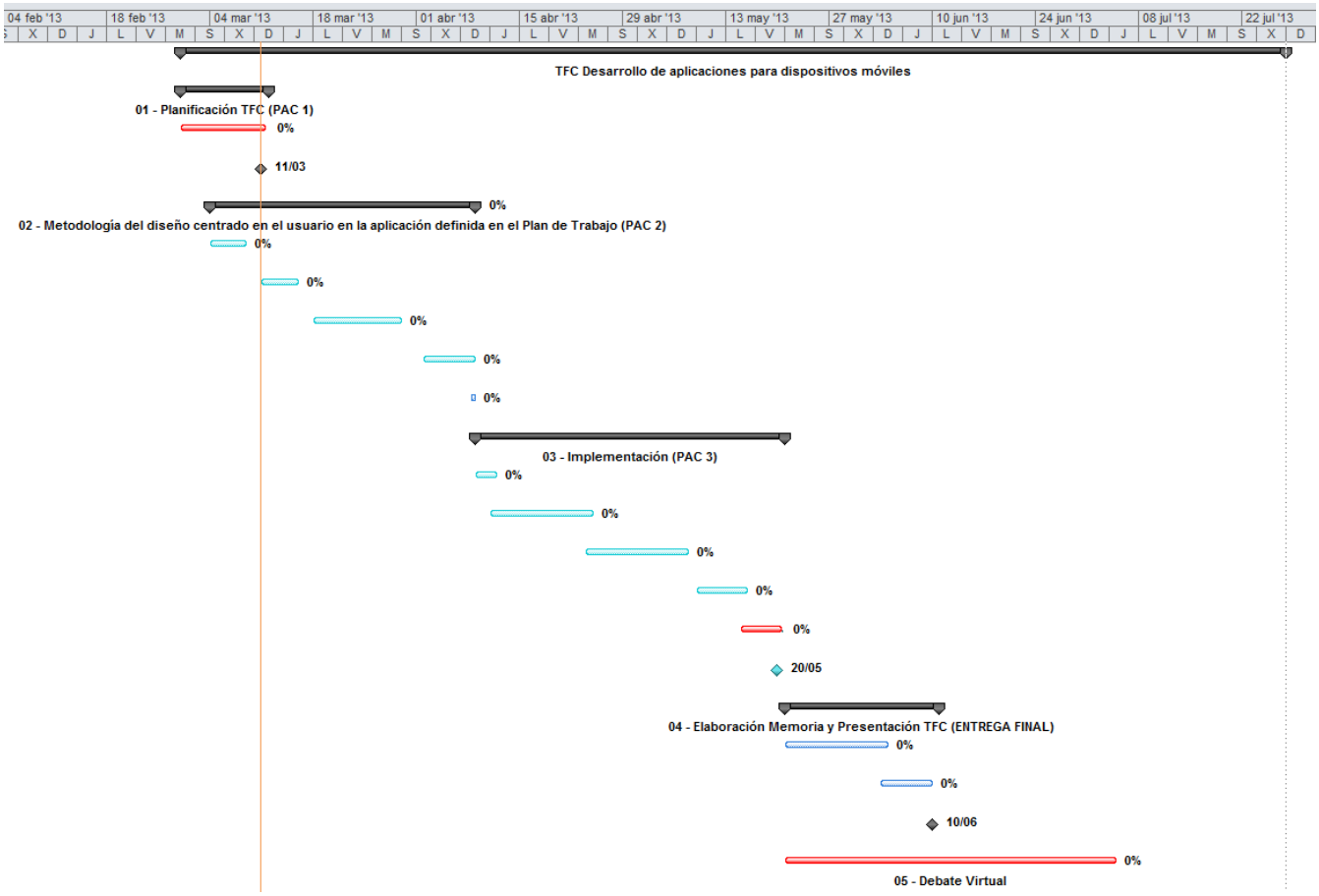
La fecha inicial del proyecto es el 28-02-2013, mientras que la fecha límite es el 10-06-2013. Disponemos de casi 108 días naturales, sin embargo, debido a mi disponibilidad de tiempo (trabajo más dos asignaturas) no me resulta posible fijar una jornada laboral de duración fija, ni designar a priori los días laborables de la semana.

Para la estimación de esfuerzos se ha considerado en general una dedicación prevista de entre 6 a 8 horas por cada semana natural del plan (7 días). No obstante, se han realizado ajustes en las duraciones de algunas tareas para tener en cuenta que su periodo de realización incluye fines de semana o festivos, ya que en esos días se pueden concentrar un mayor número de horas de trabajo.

Utilizaremos Microsoft Project 2010 para realizar la planificación. Antes, sin embargo, es necesario establecer las precedencias entre las actividades que hay que desarrollar dentro del proyecto, para lo cual utilizaremos una tabla de precedencias.

Cód. actividad	Nombre de la actividad
01	**Planificación TFC (PEC 1)**
01.01	Elaboración Plan de Trabajo TFC
01.02	Entrega Plan de Trabajo TFC (PEC 1)
02	**02 - Metodología del diseño centrado en el**
02.01	Usuarios y contexto de uso [Análisis]
02.02	Diseño conceptual [Diseño]
02.03	Prototipo [Diseño]
02.04	Evaluación [Evaluación]
02.05	Entrega PAC2
03	**Desarrollo Aplicación Generación de Datos**
03.01	Revisión de requerimientos
03.02	Construcción de las interfaces
03.03	Implementación del código
03.04	Pruebas unitarias
03.05	Pruebas de sistema y revisión
03.06	Entrega PAC 3
04	**Elaboración Memoria y Presentación TFC**
04.01	Elaboración Memoria
04.02	Elaboración Presentación TFC
04.03	Entrega Memoria y Presentación TFC

d. Diagrama de GANTT



4. Usuarios y contexto de uso

Introducción, motivación, contexto y objetivos

Lo que me ha llevado a realizar el trabajo final de carrera, el motivo y objetivo para ello es el entrar y profundizar en un ámbito profesional que es transcendente hoy en día. Las tecnologías móviles solapan la informática y tienden a convertirse y reemplazar el uso de los ordenadores personales. Esto se ha conseguido porque cada vez abarcan más posibilidades nuestros dispositivos móviles, facilitan la comunicación y el acceso a la información desde cualquier punto del planeta.

La aplicación que voy a desarrollar busca facilitar el uso de las tecnologías móviles y utilizar los recursos en nube que cada día son más amplios, y ayudan a evitar los casos de desastre por la pérdida de los datos de nuestros terminales.

Durante esta primera fase de investigación, se ha usado la técnica de *investigación contextual (contextual Enquiry)*. Esta técnica nos acerca al usuario al que observamos y preguntamos mientras realiza su trabajo o mientras interactúa con sus compañeros. De esta manera podemos entender mejor al usuario y las necesidades que puede tener.

La investigación contextual nos permite que, en etapas iniciales del proceso, realizar observaciones de la motivación del usuario a la vez que interactúa con un sistema similar al que se va a diseñar.

4.1. Investigación

a. Presentación de la investigación

Se han seleccionado varios usurarios diferentes, con perfiles laborales y de edad distintos, que nos puedan dar distintos puntos de vista sobre la aplicación y nos ayuden a elaborar una aplicación más intuitiva y útil.

Muchas aplicaciones disponen ya de servicios y aplicaciones sociales que se integran mediante *plugins* con los exploradores de imágenes, de ficheros, etc. y mediante el icono de compartir suben o publican los ficheros a estos.

Se han utilizado éstos métodos para explicarles que todo eso se concentrará en una aplicación destinada a realizar ciertas copias de seguridad de nuestros dispositivos como propósito principal.

b. Tests a usuarios.

El dispositivo sobre el que se han realizado las pruebas ha sido un móvil *Android Huawei Ideos X5* con *Android Gingerbead 2.3.5*. A los usuarios del test se les ha solicitado realizar las siguientes tareas:

- **Test 1:** Subir y compartir una foto realizada con la cámara.
- **Test 2:** Subir una foto con la App de Google Drive.
- **Test 3:** Subir una foto con la App de Dropbox.
- **Test 4:** Realizar copia de seguridad de contactos.

A continuación se muestra el resumen de los resultados de los test con información relativa a los usuarios que lo han realizado:

Usuario	Sonia, 38 años
Perfil	Administrativa
Test 1	Realizado con éxito.
Test 2	Realizado con éxito, tras breve aprendizaje.
Test 3	Realizado con éxito, tras breve aprendizaje.
Test 4	Realizado con éxito, tras breve aprendizaje.
Conclusiones	Sabe subir fotos a redes sociales, el resto no lo conocía pero tras una breve exposición realiza los tests sin problemas, apunta mejorar el acceso y que sea más intuitivito.

Usuario	Miguel, 42 años
Perfil	Técnico en informática de sistemas (Compañero de trabajo)
Test 1	Realizado con éxito.
Test 2	Realizado con éxito.
Test 3	Realizado con éxito.
Test 4	Realizado con éxito.
Conclusiones	Es conocedor de las aplicaciones y está familiarizado completamente con ellas, apunta la necesidad de aminorar pasos o sintetizar pasos.

Usuario	Daniel, 61 años
Perfil	Técnico en elevadores
Test 1	Realizado con éxito, tras breve aprendizaje.
Test 2	No realizado con éxito, tras breve aprendizaje.
Test 3	No realizado con éxito, tras breve aprendizaje.
Test 4	Realizado con éxito, tras breve aprendizaje.
Conclusiones	No es conocedor de las aplicaciones, se queja con que el tamaño de los botones e iconos son minúsculos, y que se pierde al realizar las acciones

Usuario	David, 20 años
Perfil	Estudiante de Ingeniería Industrial.
Test 1	Realizado con éxito.
Test 2	Realizado con éxito.
Test 3	Realizado con éxito.
Test 4	Realizado con éxito.
Conclusiones	Es conocedor de las aplicaciones y está familiarizado completamente con ellas, ya que las usa en su vida universitaria.

Usuario	Nerea, 17 años
Perfil	Estudiante de Bachiller
Test 1	Realizado con éxito.
Test 2	Realizado con éxito.
Test 3	Realizado con éxito.
Test 4	Realizado con éxito.
Conclusiones	No es conocedora de la totalidad de las aplicaciones, sin explicación alguna ha realizado todos los test, se nota que está familiarizada con el uso diario del terminal.

c. Conclusiones.

Tras la investigación y test realizados se concluye que en los rangos de usuarios de edades comprendidas entre 15 a 45 años realizan de forma exitosa la totalidad de los test siendo necesario un breve aprendizaje dependiendo del caso.

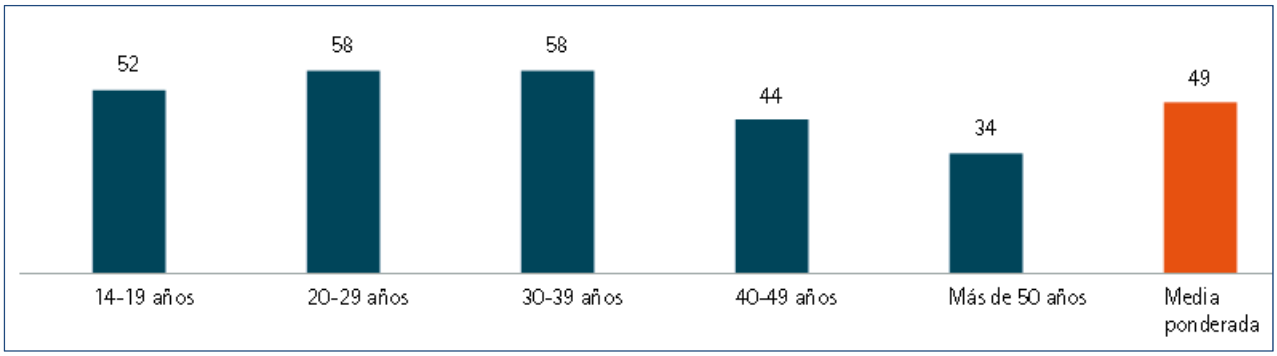
Algunas de las ideas y apuntes realizados sobre las pruebas han sido:

- **Simplificar y reducir acciones**, es decir, que con un mínimo de pulsaciones y cambios de pantalla realice aquella acción de forma exitosa.
- **Aumentar el tamaño y la simplicidad de los iconos**, pues en los perfiles de edad superior o con problemas de visibilidad ayuda a realizar de forma intuitiva las tareas.
- **Simplicidad y minimizar cantidad de colorido**, ya que con una gama de colores mucho más básica facilita la visibilidad de los iconos, y ayuda a diferenciar claramente cada acción.

4.2. Perfiles de usuario.

Tras aplicar los métodos de observación e investigación conceptual en campo, y buscar observaciones y otros estudios en la red, como por ejemplo el realizado por la empresa multinacional *Accenture* que expone que más del 60% de los usuarios que acceden a Internet mediante sus Smartphones lo hacen a diario. Los usuarios más activos se encuentran en el rango de edades comprendidas entre los 14 y los 49 años. Los ratios de adopción por intervalos de edad son los siguientes:

Ratio de uso de internet móvil por edades.



Fuente:http://www.accenture.com/SiteCollectionDocuments/Local_Spain/PDF/Accenture_Ametic_2011_Retos_y_oportunidades.pdf

Con lo cual se puede deducir que trabajaremos con una amplitud de perfiles de edad definidos:

- **De 15 a 25 años**, familiarizados con la tecnología móvil, y acostumbrados a trabajar con aplicaciones que interactúan con servicios de red, cambian de móvil como mínimo en un plazo de dos a tres años. Estudios hechos demuestran que más de un 70% de los jóvenes de esta edad ya poseen un terminal Smartphone y lo utilizan ampliamente.
- De 25 a 50 años, acostumbrados a los cambios de generación, sin la rapidez de aprendizaje del anterior grupo pero se integran ampliamente con los servicios y aplicaciones de red móviles. En este grupo el uso de tecnologías móviles de última generación no es tan amplia como en el grupo anterior,
- Mayores de 50 años, grupo que más problemas tiene para el uso de las nuevas tecnologías, reticentes a los cambios y evoluciones constantes de las telecomunicaciones, usan casi forzosamente dispositivos móviles de última generación por la desintegración de los sistemas móviles de primera generación.

No abarcamos edades inferiores a 15 años, aunque empiezan a aparecer grupos de edades entre los 9 a 14 años con cada vez mayor posesión de un terminal móvil con aplicaciones de mensajería instantánea a través de redes de datos. Tampoco abarcamos mayores de 60 porque hay un porcentaje mínimo de éstos de uso de esta tecnología.

4.3. Contextos de uso

Los contextos de usos atenderán principalmente a los factores que nuestra aplicación influirá en su uso y en el grado de satisfacción conseguida por parte de unos usuarios específicos, tenemos que llegar a comprender las circunstancias en las cuales va a ser usado.

El contexto de uso principal será aquel en el que un usuario cualquiera, en su *"qué hacer"* diario, y por las fatalidades de la volatilidad temporal de los datos en la tecnología, ya sean fallos de software, de hardware o mal uso por parte del usuario, éste pierde parte o la totalidad del contenido de su dispositivo móvil (fotos, contactos, carpetas de ficheros, etc.), si no se toman ciertas precauciones antes el resultado puede ser traumático para el usuario.

Ante tal *"fatalidad"*, el usuario precavido que posee la aplicación y ha realizado en ésta las copias de seguridad de sus contactos, de las fotos de la carpeta de su cámara (las cuales son recuerdos que no quiere perder y valora en gran medida) y de varios ficheros puntuales, el usuario solamente está preocupado por recuperar su terminal o por el tener que desembolsar el coste de tener que adquirir uno nuevo.

4.4. Análisis de las tareas

Entre las numerosas tareas que los usuarios pueden y realizar, se encuentran las siguientes:

- **Tarea 1 – Copy2Cloud:** Hacer foto con la cámara y subirla al servicio de nube.

  1. Pulsar sobre el icono de la tarea.
  2. Nos solicita la cuenta de usuario del servicio de nube que se quiere utilizar.
  3. Abre el dispositivo de la cámara y se realiza la foto.
  4. Guarda la foto en el servicio de nube.

- **Tarea 2 – Files2Cloud:** Consultar un punto existente

  1. Pulsar sobre el icono de la tarea.
  2. Seleccionamos subir ficheros o bajar ficheros.
  3. Seleccionamos el/los fichero/s o carpetas a subir/bajar en el explorador de ficheros.
  4. Pulsamos que realice la acción de lo seleccionado.
  5. Nos solicita la cuenta de usuario del servicio de nube que se quiere utilizar.
  6. Guarda la selección en el servicio de nube/el terminal móvil.

- **Tarea 3 – Write2Cloud:** Editar notas sobre la nube.

  1. Pulsar sobre el icono de la tarea.
  2. Nos solicita la cuenta de usuario del servicio de nube que se quiere utilizar.
  3. Nos abre el explorador de ficheros filtrado por documentos de notas.
  4. Seleccionamos el fichero a editar o creamos uno nuevo.

- **Tarea 4 – Play2Cloud:** Escuchar música de ficheros subidos en el servicio de la nube.

  1. Pulsar sobre el icono de la tarea.
  2. Nos solicita la cuenta de usuario del servicio de nube que se quiere utilizar.
  3. Reproduce la lista de ficheros seleccionados.

- **Tarea 5 – Contacts2Cloud:** Guarda o restaura el fichero de contactos.

  1. Pulsar sobre el icono de la tarea.
  2. Nos solicita la cuenta de usuario del servicio de nube que se quiere utilizar.
  3. Seleccionamos si queremos guardar o recuperar la copia de los contactos.
  4. Realiza la acción seleccionada.

- **Tarea 6 – Sync2Cloud:** Escuchar música de ficheros subidos en el servicio de la nube.

  1. Pulsar sobre el icono de la tarea.
  2. Nos solicita la cuenta de usuario del servicio de nube que se quiere utilizar.
  3. Se seleccionan en el explorador las carpetas o ficheros que queremos tener sincronizados.
  4. Se acepta la tarea y automáticamente pasa a sincronizar.

5. Diseño conceptual

a. Escenarios de uso

Como bien sabemos, un escenario de uso describirá desde el punto de vista del usuario, como se aplicará nuestra utilidad en una situación concreta, esto ayudará y contribuirá al desarrollo de nuestra aplicación, determinará necesidades de los usuarios y de diseño.

Se pueden definir diversidad de escenarios, ya que la polivalencia de la aplicación dará lugar a poder ser usada en diferentes contextos. Aunque la finalidad principal es la de realizar copias de seguridad, también da la posibilidad de redactar notas, obtener fotos de la cámara y escuchar música almacenada en la nube, lo cual generará múltiples posibilidades.

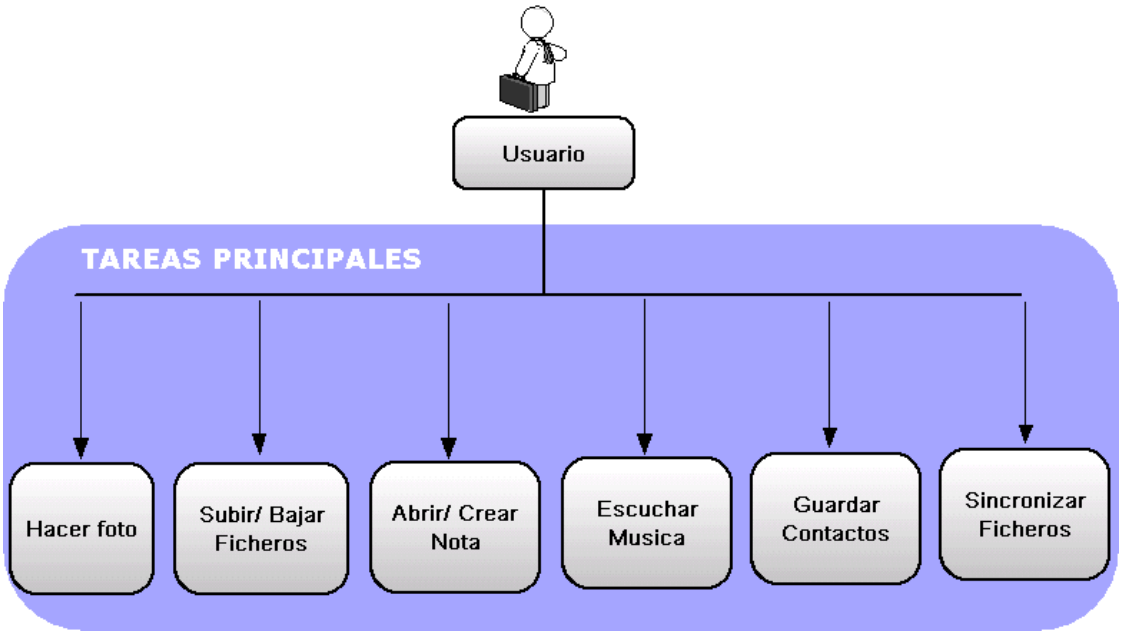
Un escenario sería aquel en el que un usuario de la calle cualquiera, con un móvil multimedia de última generación y con una cámara de 8 megapíxeles, estando de viaje tiene la oportunidad de realizar con su dispositivo móvil unas fotos formidables las cuales no quiere perder, y necesita tener con toda seguridad una copia de respaldo. Este usuario utilizaría la tarea de realizar fotos directamente en la nube y así no solo tendría la seguridad de no perderlas si no que desde cualquier otro dispositivo puede acceder a ellas y mostrarlas a familia y amigos tras la vuelta de su viaje.

Otro escenario podría ser un estudiante universitario, que tiene que estar tomando constantes notas en clase, almacenar ficheros de documentos de apuntes y en sus ratos libres escuchar música que por la gran cantidad de ficheros mp3 almacenados los tiene en el servicio de nube ordenado por carpetas. Este usuario usará la aplicación para todos estos fines e incluso utilizará casi la totalidad de las tareas.

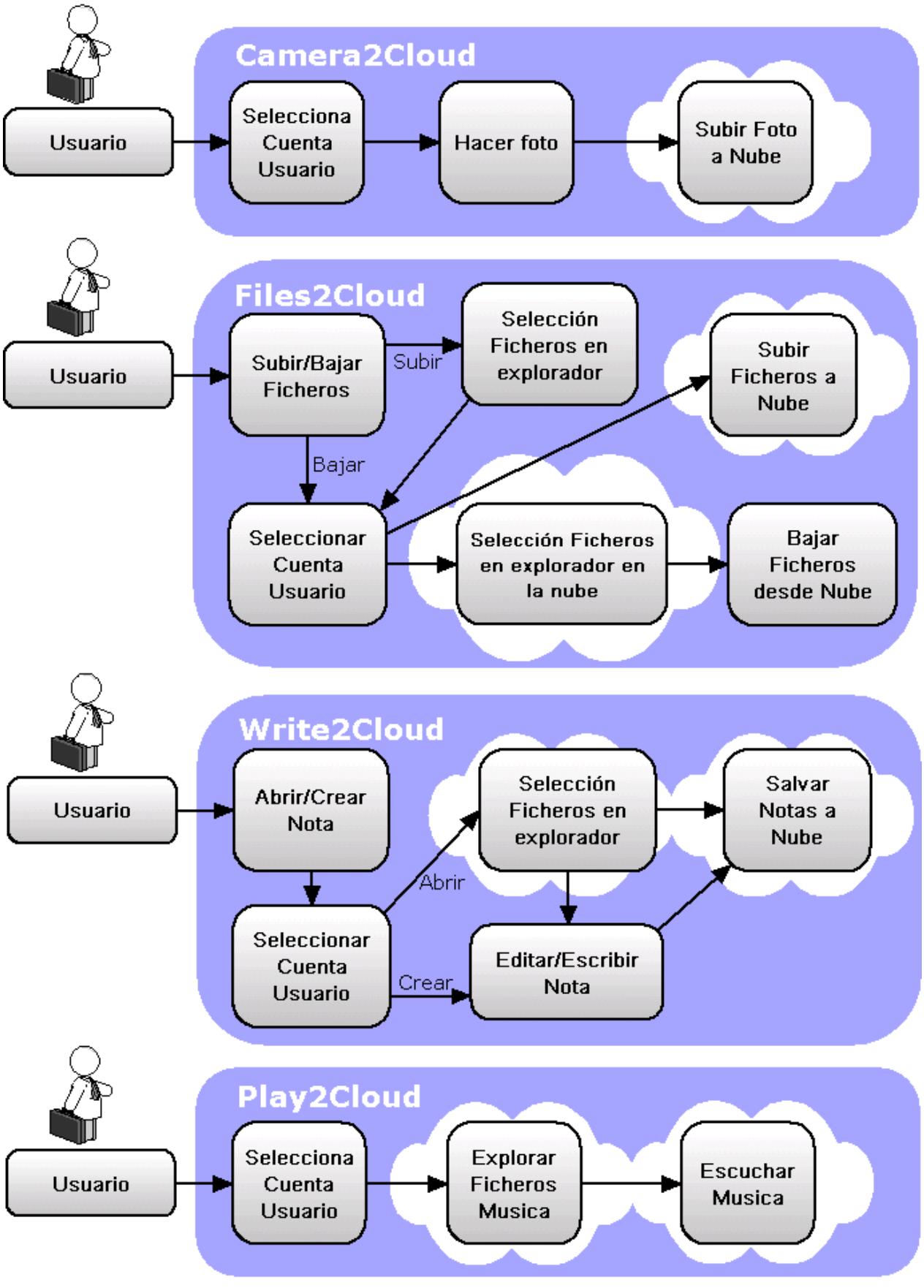
Un tercer escenario podría ser en un entorno laboral empresarial, en el que un ejecutivo o un comercial con una gran agenda de contactos (teléfonos, números de faxes, correos electrónicos, direcciones, etc.), la cual considera una herramienta esencial de su trabajo. Con lo cual quiere tener copias de seguridad de todo ello y poder acceder a ella en caso de un hecho catastrófico en su terminal móvil. También usara las anotaciones que podrá ver tanto desde su terminal móvil como su equipo de escritorio.

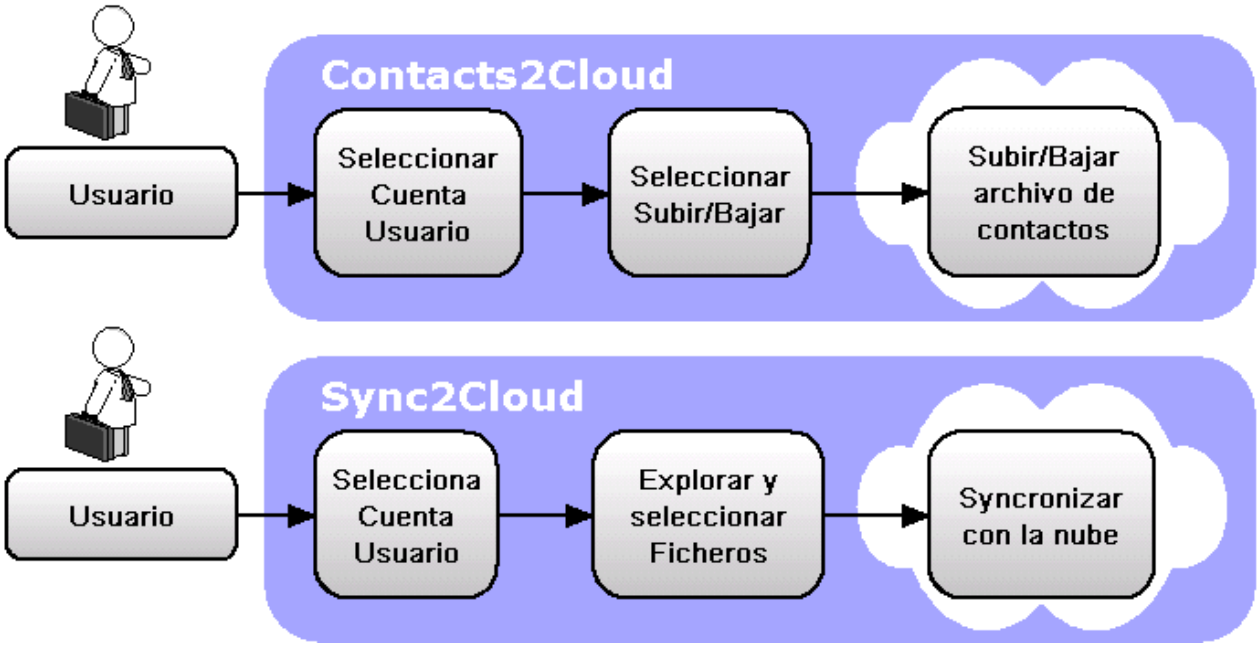
b. Diagramas de casos de uso

En este punto, tras haber definido los escenarios, se describen los diferentes flujos o estados por los que pasa la aplicación en su proceso de interacción con el usuario. Primeramente, el primer diagrama de flujos esquematiza las tareas principales que puede desempeñar el usuario en la aplicación.



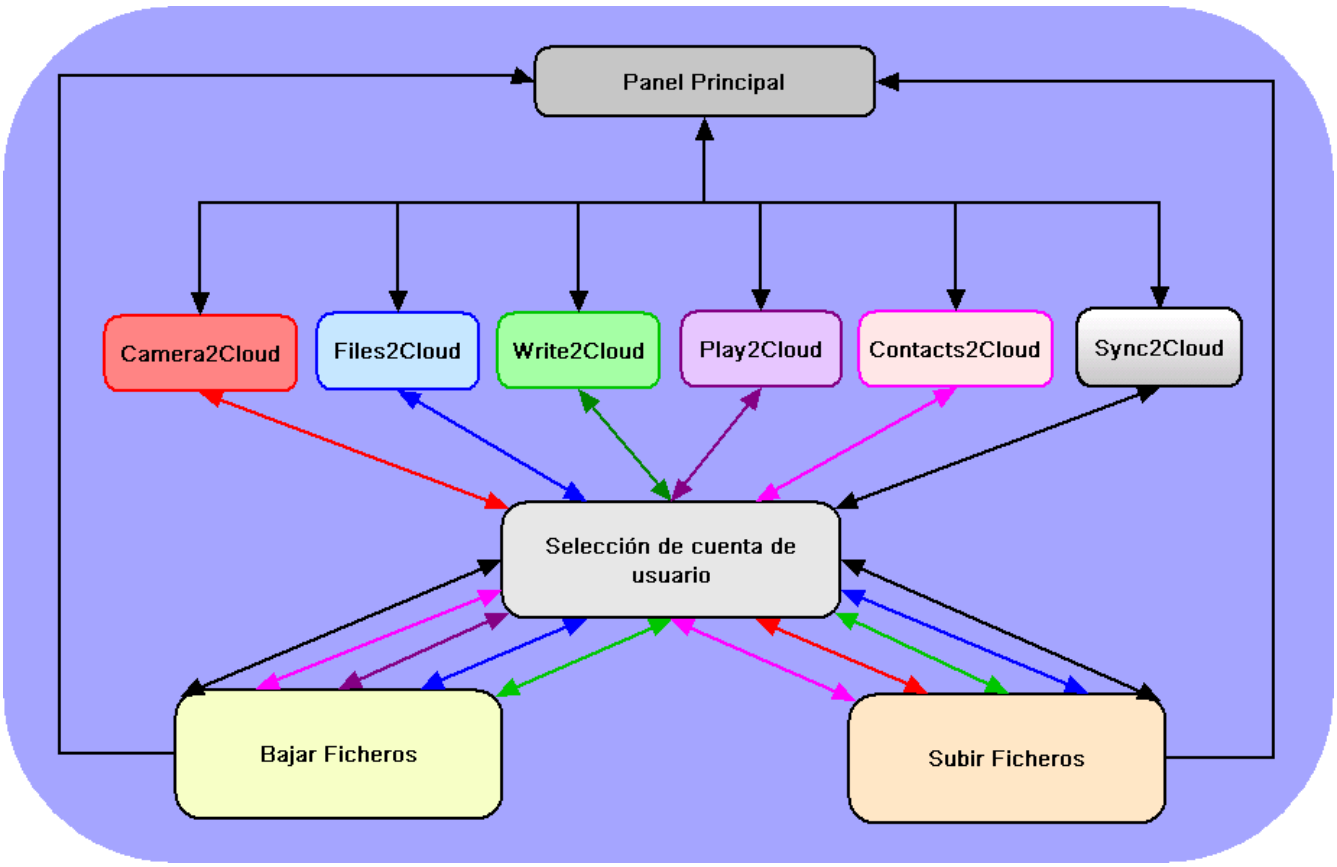
Los siguientes diagramas de casos de uso, esboza por separado, cada una de las tareas que puede realizar el usuario.





c. Diagramas de estados. Flujos de interacción

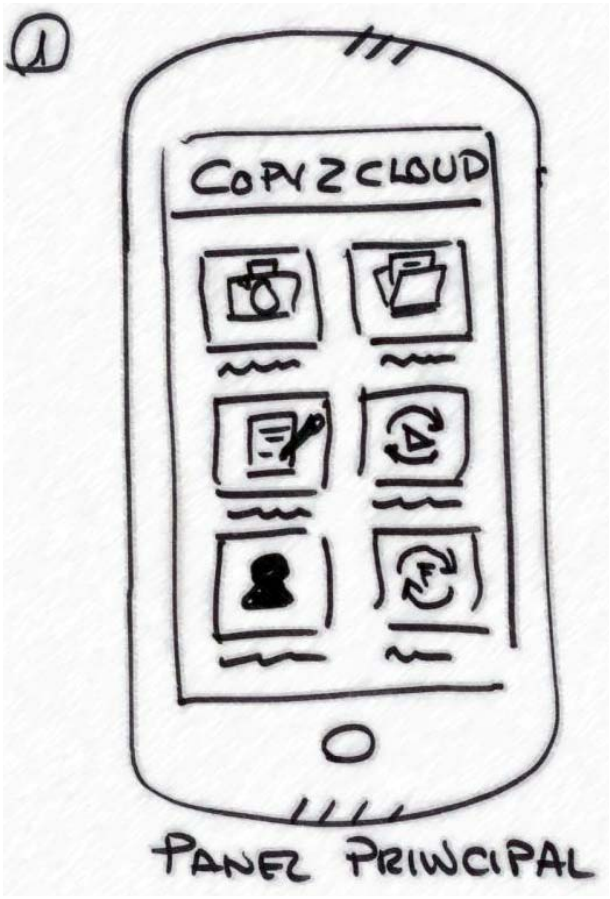
A partir de este punto, se describe los diferentes flujos o estados por los que pasa la aplicación en su proceso de interacción con el usuario. Simplificando los procesos, la aplicación moverá diferentes tipos y formatos de ficheros entre el dispositivo móvil y el servicio de ficheros en nube, a partir del panel principal, linealmente se suceden las acciones entre las actividades, con posibilidad siempre de volver atrás.



6. Prototipo

6.1. Sketches

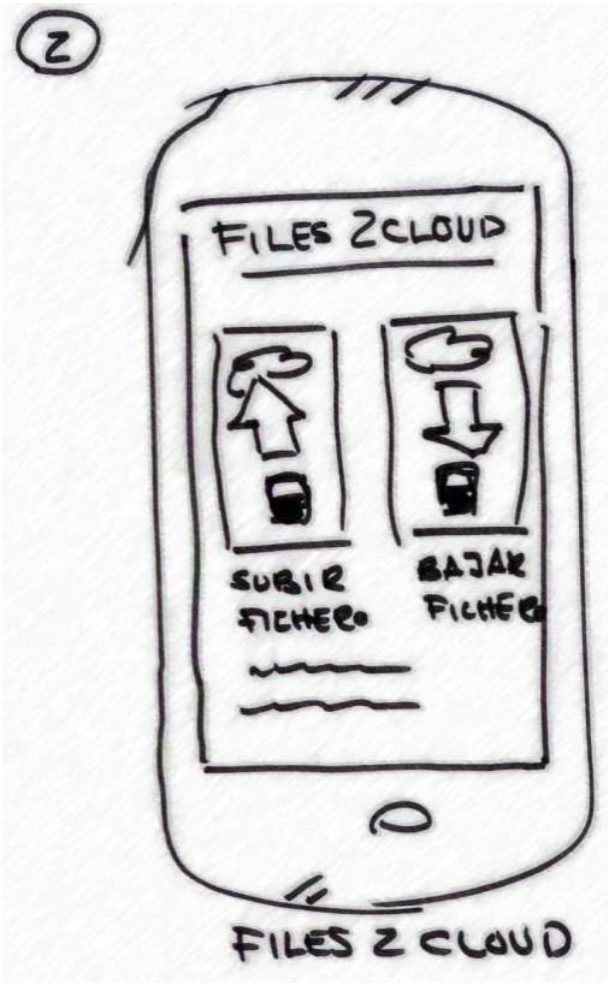
Se han realizado varios sketches que nos han ayudado a orientarnos mejor en el diseño final de la aplicación.



En el primer sketch, se realiza el primer esbozo de lo que será el "Panel Principal" (Dashboard), es la interfaz de inicio de *Copy2Cloud* que se comprende las seis actividades principales:

- Camera2Cloud
- Files2Cloud
- Write2Cloud
- Play2Cloud
- Contacts2Cloud
- Sync2Cloud.

Los botones serán de gran tamaño, facilitando la identificación rápida de las tareas y la pulsación.

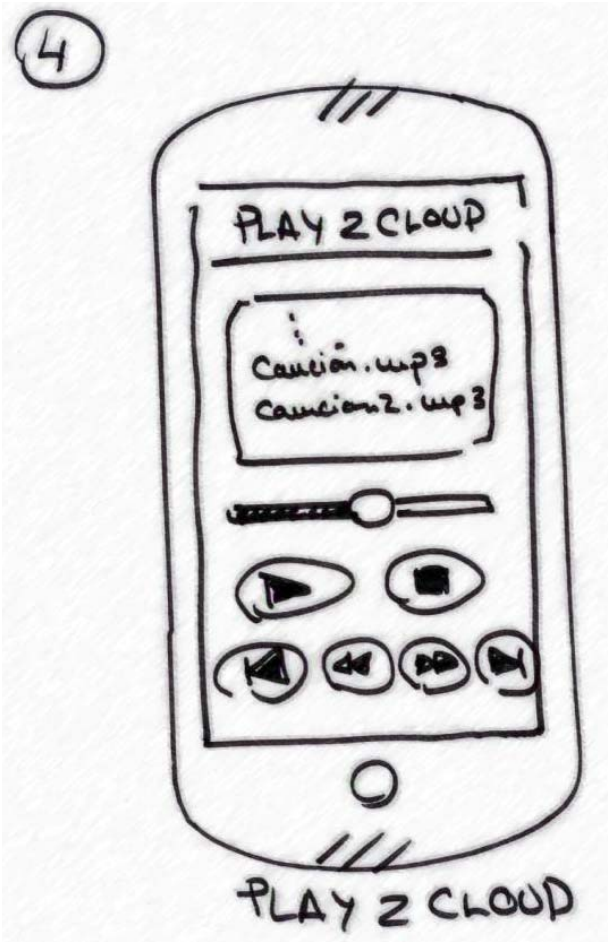


En el segundo sketch se muestra una de las principales tareas de la aplicación: *Files2Cloud*, la cual permite la navegación tanto en la nube como en el terminal móvil, e intercambiar ficheros entre ambas plataformas de almacenamiento.

Este panel es de gran simplicidad, solamente posee en el inicio dos botones de gran tamaño, uno especifica si lo que se desea es subir ficheros y el otro lo contrario, si se desea bajar ficheros al terminal móvil. Se mostrarán conforme se vayan seleccionando un listado de los ficheros que se han ido marcando.

El tercer sketch, muestra el acceso a la cámara desde la tarea *Camera2Cloud*, esta actividad es la que visualmente es la más sencilla de todas, no tiene panel gráfico, simplemente da acceso a la cámara del teléfono móvil, realiza la foto y directamente pasa a subirla al servicio de nube. Una vez termine de realizar la foto se mostrará un mensaje de fichero subido que verificará la acción realizada.

Si no gusta la foto, se podrá repetir, mientras no se afirme el guardado de la foto, la aplicación estará en espera para subir el fichero de imagen al servicio de almacenamiento de ficheros en red.



El cuarto sketch, es la tarea *Play2Cloud*, la cual es un simple reproductor de la música que tenemos almacenada en la nube.

Este panel posee un listado de las canciones que se va a reproducir, una línea de tiempo, botones de play, stop, pause y aquellos que permiten el paso de una canción a otra.

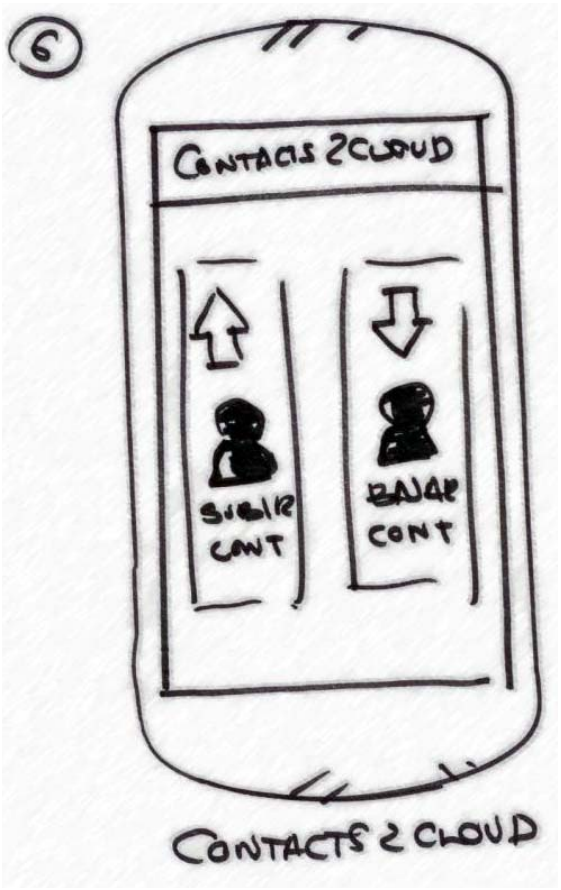
El listado se confeccionará desde un explorador de ficheros en el servicio de ficheros en la red.

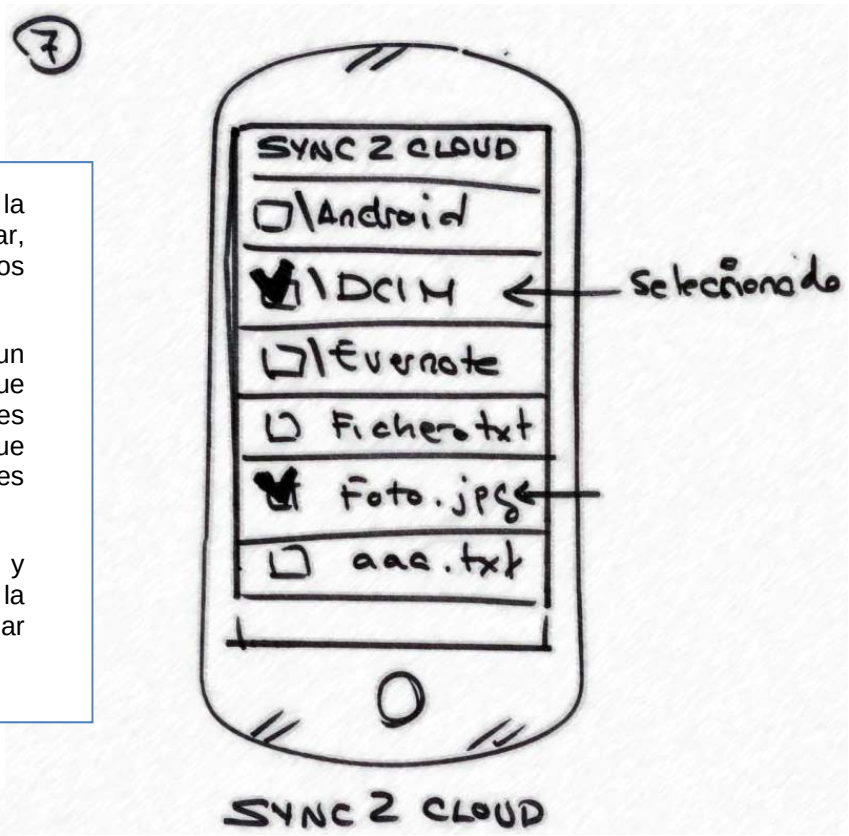
El quinto sketch, nos muestra el panel de escritura de notas de Write2Cloud.

En esta tarea se abrirá directamente con la nota vacía, para guardarla a la nube, si le damos al botón de menú nos mostrará un panel en la parte inferior de la actividad que nos dará la opción de abrir un fichero en la red, salvar el que estamos editando y cerrar la tarea.

Cuando queramos elegir un fichero de la nube, saldrá un explorador que filtrará y mostrará solamente los ficheros de notas.

El sexto sketch muestra un panel bastante básico y sencillo, muy parecido al panel inicial de *Files2Cloud*, éste posee solamente dos botones de gran tamaño y visibilidad que comprenden las funciones básicas de esta tarea. Una es la de realizar una copia de los contactos de terminal móvil y subirlos a la red, y la otra hace justo lo contrario, selecciona el fichero desde la red y lo restablece en el terminal.

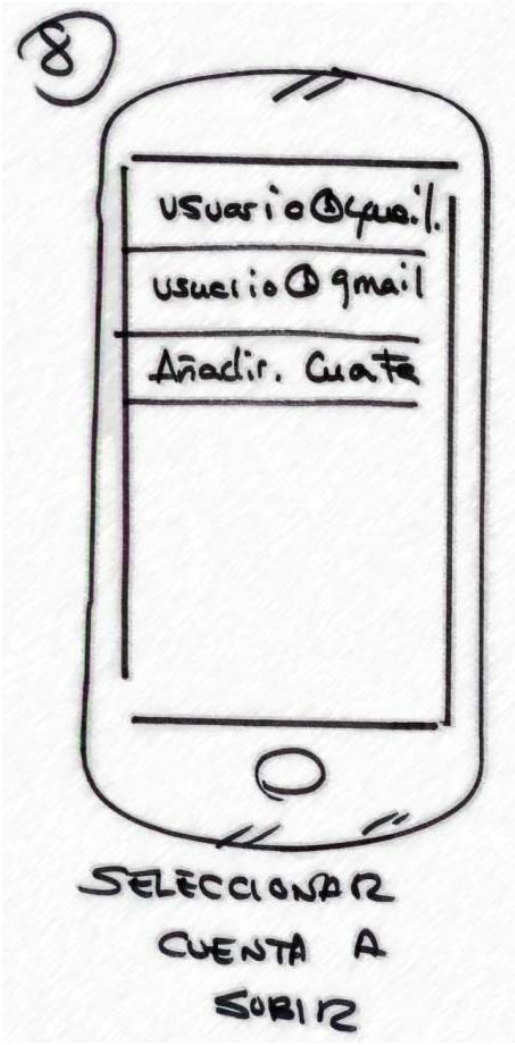




El sketch número siete muestra la última tarea que nos queda por explicar, este es el de sincronización de ficheros con el servicio de nube.

Este también posee únicamente un explorador de ficheros locales que permitirá seleccionar con marcadores aquellos ficheros y carpetas que deseamos que estén en ambos lugares (terminal y nube).

La sincronización buscará y comparará el número y de ficheros y la fecha de modificación para almacenar siempre el que tenga la más reciente.



Este último y octavo sketch, comprende el panel que saldrá en todas las tareas en su momento oportuno, como sabemos los terminales móviles permiten tener varias cuentas de usuario, y en el momento que aparezca esa pantalla será cuando nos pedirá la cuenta del usuario en la que deseamos hacer esa acción determinada.

Este panel muestra un listado de los usuarios de correo que están registrados en el terminal o si deseamos añadir uno nuevo y realizar la acción con el elegido.

6.2. Prototipo horizontal de alta fidelidad

Apoyándonos en los pasos dados sobre el análisis de usuarios, contextos de uso, diseño conceptual y sketches, realizamos el siguiente prototipo, donde apreciaremos de forma horizontal el funcionamiento de la aplicación en sus diferentes pantallas.

- **Panel Principal:** Es la pantalla de inicio de la aplicación, presenta una botonera que dará paso a las tareas principales. Se utilizan colores sencillos y formas claras, como habíamos planificado en los sketches inicialmente.

  Contiene las seis tareas principales: *Camera2Cloud*, *Files2Cloud*, *Write2Cloud*, *Play2Cloud*, *Contacts2Cloud* y *Sync2Cloud*. Cada una de ellas tiene un icono amplio, que abarca gran parte del botón y claramente distinguible.

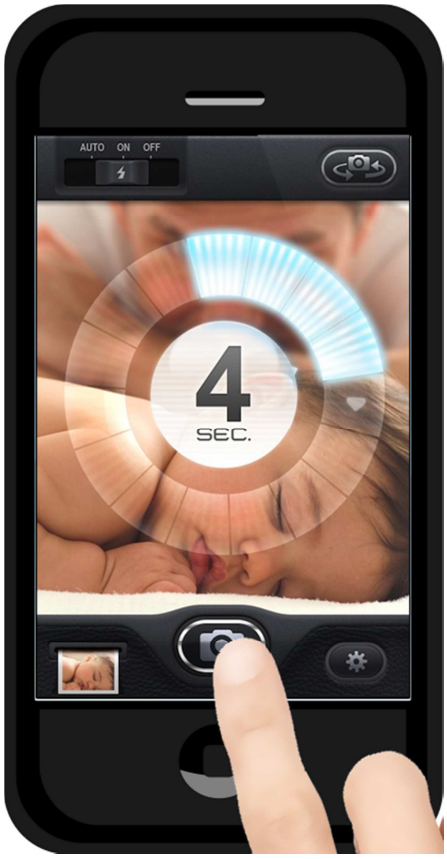
Si pulsamos el botón de menú, aparecerá un submenú con pestañas, en el que saldrán las siguientes acciones:

- **Settings:** mostrará una pantalla de propiedades y configuraciones de la aplicación base.
- **About:** contiene la información de la aplicación: Desarrollador, fecha de publicación, número de versión, etc.
- **Close:** Cierra la aplicación y vuelve al sistema operativo.

Camera2Cloud

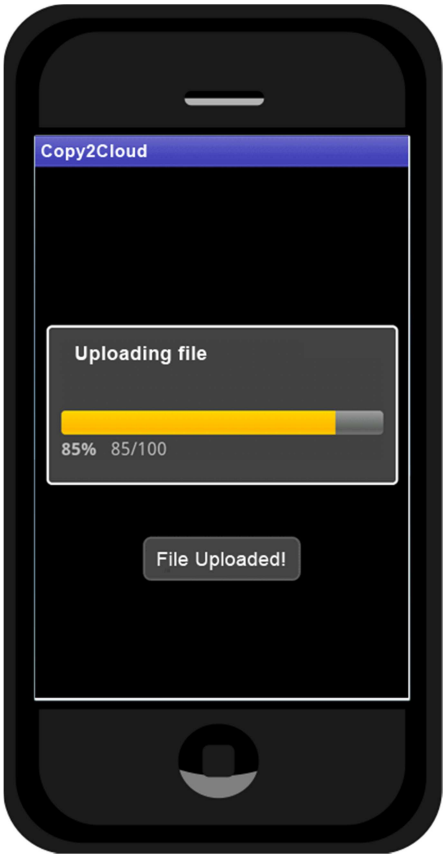
La tarea de *Camera2Cloud,* como ya habíamos comentado realizará fotos y las subirá a la red. Nada más pulsar sobre su icono nos enseñará un panel en el que seleccionaremos la cuenta de usuario a la que va a ser subida la foto una vez realizada.

También hay posibilidad desde el panel añadir una cuenta nueva, posee abajo un par de botones de confirmación y cancelación de la acción que nos devolvería al panel principal.



Al proseguir con la confirmación, arrancara la cámara predeterminada o nos deja seleccionar entre aquellas que tengamos instaladas.

Una vez se realiza la foto, al confirmar pasará a otra pantalla, en la que se realizará la subida del fichero y nos enunciará si se ha producido con éxito, entonces volverá al panel principal de la aplicación.

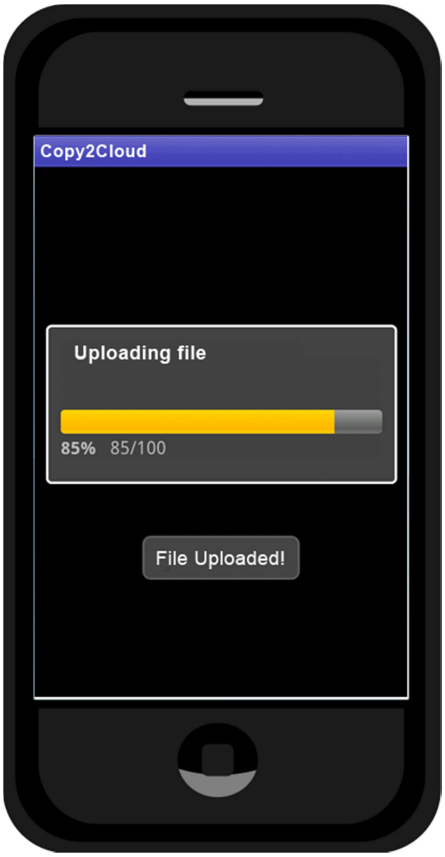


Files2Cloud

Escena 1

Escena 2

Escena 3



Escena 4

Files2Cloud, comienza con (escena 1) un par de iconos de gran tamaño que diferencia la acción que se han de realizar:

- Subir ficheros.
- Bajar ficheros.

Al entrar a realizar una de estas acciones, aparecerá un nuevo panel (escena 2) en el que están los iconos de añadir fichero y realizar la acción (subir ficheros/ descargar ficheros).

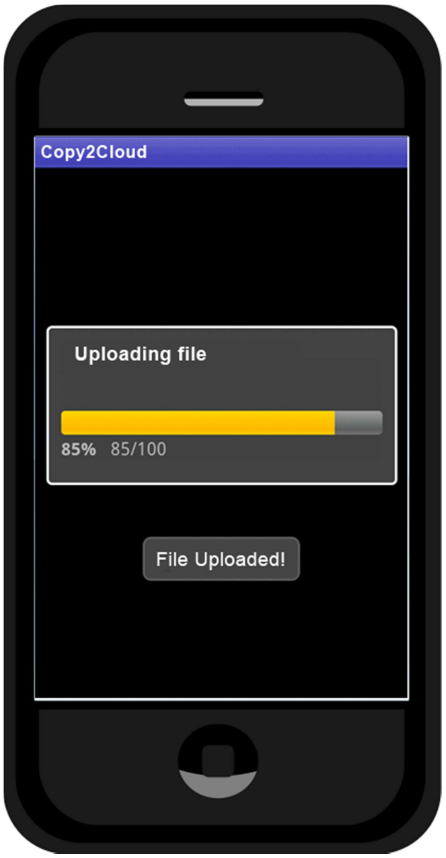
Si pulsamos el primero aparece la siguiente pantalla con un explorador de ficheros (escena 3) en el que tras elegir uno en concreto, volverá al panel anterior conforme vayamos seleccionando se irán añadiendo al listado de ficheros del anterior panel (escena 2).

Para finalizar se pulsará el icono de subir/bajar fichero (escena 2) y se realizará la acción deseada (escena 4).

Write2Cloud

La siguiente tarea es Write*2Cloud,* el cual muestra un editor de notas en blanco, en el que podemos escribir directamente.

Una vez escrito aquello que se desea anotar, dando al botón de menú, saldrá un submenú flotante que muestra tres acciones: Abrir nota, salvar nota y cerrar.



Si salvamos la nota nos pedirá el nombre de fichero con el que queremos guardarla y la subirá directamente a la red.

Si elegimos abrir una nos mostrará en el navegador filtrado por ficheros de texto aquellos que hay en la nube.

Si por último elegimos cerrar volveremos al panel principal.

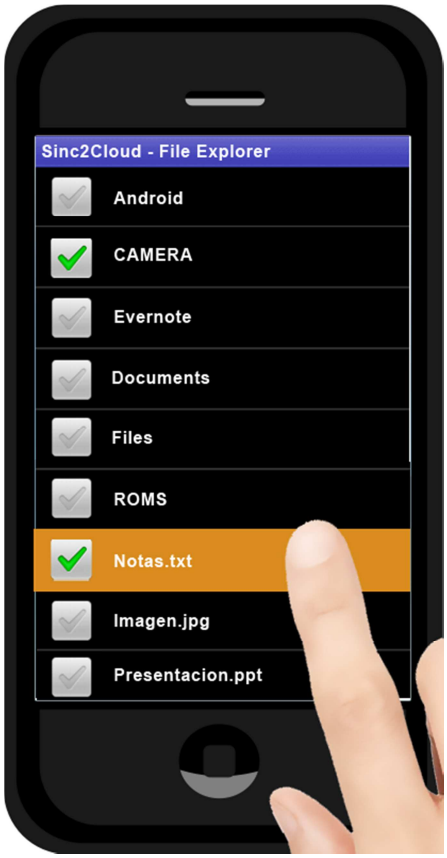
Play2Cloud

La cuarta tarea es *Play2Cloud*, ésta tiene un panel muy simple, con un listado de las canciones seleccionadas, una línea de tiempo y los botones de operación de la reproducción.

La aplicación tendrá un submenú parecido a los descritos en *Write2Cloud*, en los que abrirá un explorador de ficheros con filtro para los ficheros mp3, y conforme se vayan seleccionando se añadirán a la lista de canciones.

En el listado de aplicaciones si se seleccionan mientras la reproducción está detenida permitirá eliminar esa canción del listado.

Sync2Cloud



La quinta tarea es Sync2Cloud, primeramente abrirá un explorador más completo que el visto en *Files2Cloud*, en el que seleccionaremos aquellos ficheros y carpetas que creemos que son indispensables y se desea sincronizar con la nube.

Dispondrá de un submenú que se podrán hacer las siguientes tareas:

- **Validar:** guardará la selección realizada y comenzará a sincronizar.
- **Limpiar selección:** Limpiara los campos seleccionados.
- **Cerrar:** Cierra la tarea y vuelve al panel principal.

Contacts2Cloud

La última tarea que se especifica en el proptotipado es *Contacts2Cloud*.

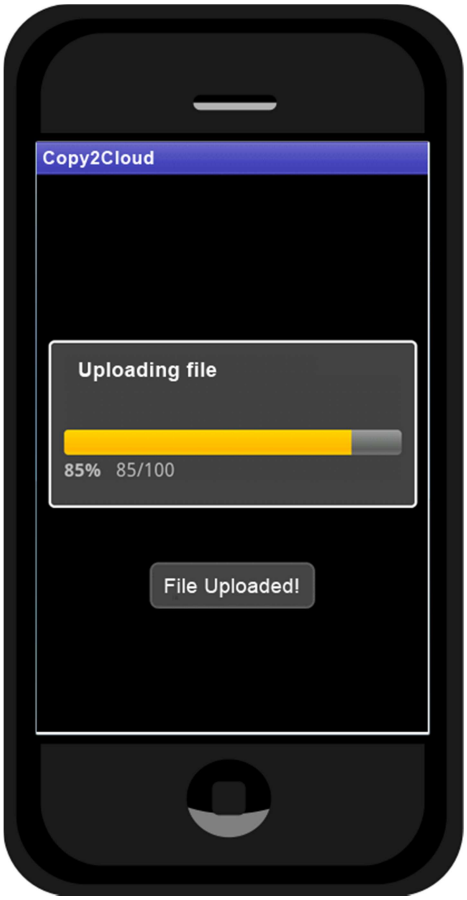
Esta tarea realiza una copia del fichero de contactos con extensión "*vcf*", y lo sube directamente a la red.

El panel principal de la tarea tiene dos botones de gran tamaño:

- **Upload Contacts File:** Nada más pulsarlo, realizará la copia del fichero de contactos y procederá a subirlo.
- **Downoad Contacts File:** Nada más pulsarlo, descargará de la red el fichero de contactos y restaurará la copia.

Como en todas las tareas se seleccionará la cuenta en la que se subirá el fichero de contactos y procederá a subirlo.

Se mostrará en todo momento con barras de progreso y elementos de aviso (Toast) para tener informado de que se han producido con éxito el volcado en cada plataforma, tanto de red como en el terminal móvil.



7. Evaluación – Test de usuario

7.1. Pack de preguntas a los usuarios que realizarán el test.

Como sabemos el proceso de DCU es iterativo y redundante, y como consecuencia tenemos que evaluar y reevaluar los diseños corrigiéndolos y perfeccionándolos.

Procedemos a definir las preguntas que ayudarán a conocer opiniones de los usuarios respecto al diseño centrado en el usuario de nuestra aplicación.

- ¿Cree que es intuitivo el panel principal?
- ¿Añadiría algo? ¿Quitaría algo? ¿Modificaría algo?
- ¿Sabe lo que es un servicio de ficheros online del tipo nube?
- ¿La sucesión en el desarrollo de las tareas cree que es la adecuada?
- ¿La iconografía es comprensible?
- ¿Cree que son muchas pantallas a la hora de realizar las tareas?
- ¿Cree que la velocidad de acceso y navegación es aceptable?
- ¿Cree que son de utilidad todas las tareas?
- ¿Cree que las podrá utilizar en cualquier momento?

7.2. Tareas que han de realizar los usuarios

Los usuarios del test de *Copy2Cloud* se les requerirán realizar la siguiente lista de tareas:

- Realizar una foto y subirla a la red.
- Realizar una subida de un fichero al servicio de nube en red y luego vuélvalo a descargar.
- Crear una nota, editarla, guárdala en la red y luego recuperar
- Subir un fichero o varios de música en formato mp3 al servicio de red y reproduzca una selección desde la aplicación.
- Realizar una copia de seguridad de los contactos, guardarla en la red y posteriormente restaurarla.
- Realizar una tarea de sincronización de ciertos ficheros de su terminal y verificar que se ha realizado. - Rellenar los campos solicitados.

-

7.3. Preguntas referentes a las tareas

A los usuarios de Copy2Cloud se les realizarán las siguientes preguntas respecto a las tareas realizadas:

- ¿Ha encontrado dificultades o trabas para completar las tareas con éxito?
- ¿Qué elementos cree que carece el prototipo en fase beta actual?
- ¿Ha encontrado algún fallo o defecto mientras usaba el prototipo?

- ¿Cree que podría mejorarse la herramienta?
- ¿Cree que la aplicación es intuitiva?
- ¿Cumple con sus expectativas?
- Tarea Camera2Cloud ¿Ha realizado la tarea con éxito?
- Tarea Camera2Cloud ¿Le ha resultado sencillo e intuitivo?
- Tarea Camera2Cloud ¿Cree que es de utilidad y la usará normalmente?
- Tarea Files2Cloud ¿Ha realizado la tarea con éxito?
- Tarea Files2Cloud ¿Le ha resultado sencillo e intuitivo?
- Tarea Files2Cloud ¿Cree que es de utilidad y la usará normalmente?
- Tarea Write2Cloud ¿Ha realizado la tarea con éxito?
- Tarea Write2Cloud ¿Le ha resultado sencillo e intuitivo?
- Tarea Write2Cloud ¿Cree que es de utilidad y la usará normalmente?
- Tarea Play2Cloud ¿Ha realizado la tarea con éxito?
- Tarea Play2Cloud ¿Le ha resultado sencillo e intuitivo?
- Tarea Play2Cloud ¿Cree que es de utilidad y la usará normalmente?
- Tarea Contacts2Cloud ¿Ha realizado la tarea con éxito?
- Tarea Contacts2Cloud ¿Le ha resultado sencillo e intuitivo?
- Tarea Contacts2Cloud ¿Cree que es de utilidad y la usará normalmente?
- Tarea Sync2Cloud ¿Ha realizado la tarea con éxito?
- Tarea Sync2Cloud ¿Le ha resultado sencillo e intuitivo?
- Tarea Sync2Cloud ¿Cree que es de utilidad y la usará normalmente?

8. Implementación

8.1. Arquitectura de la aplicación

Tras haber realizado la planificación y el prototipo de nuestro proyecto, comenzamos la puesta en marcha e implementación del código que realizará todas las funciones que pretendemos que realice la aplicación Copy2Cloud.

La aplicación consta de dos capas claramente diferenciadas, la aplicación cliente Copy2Cloud que se ejecuta en un dispositivo Android y la parte servidora, los servicios de ficheros en red, dónde se alojarán los datos que deseemos mover o crear desde la aplicación.

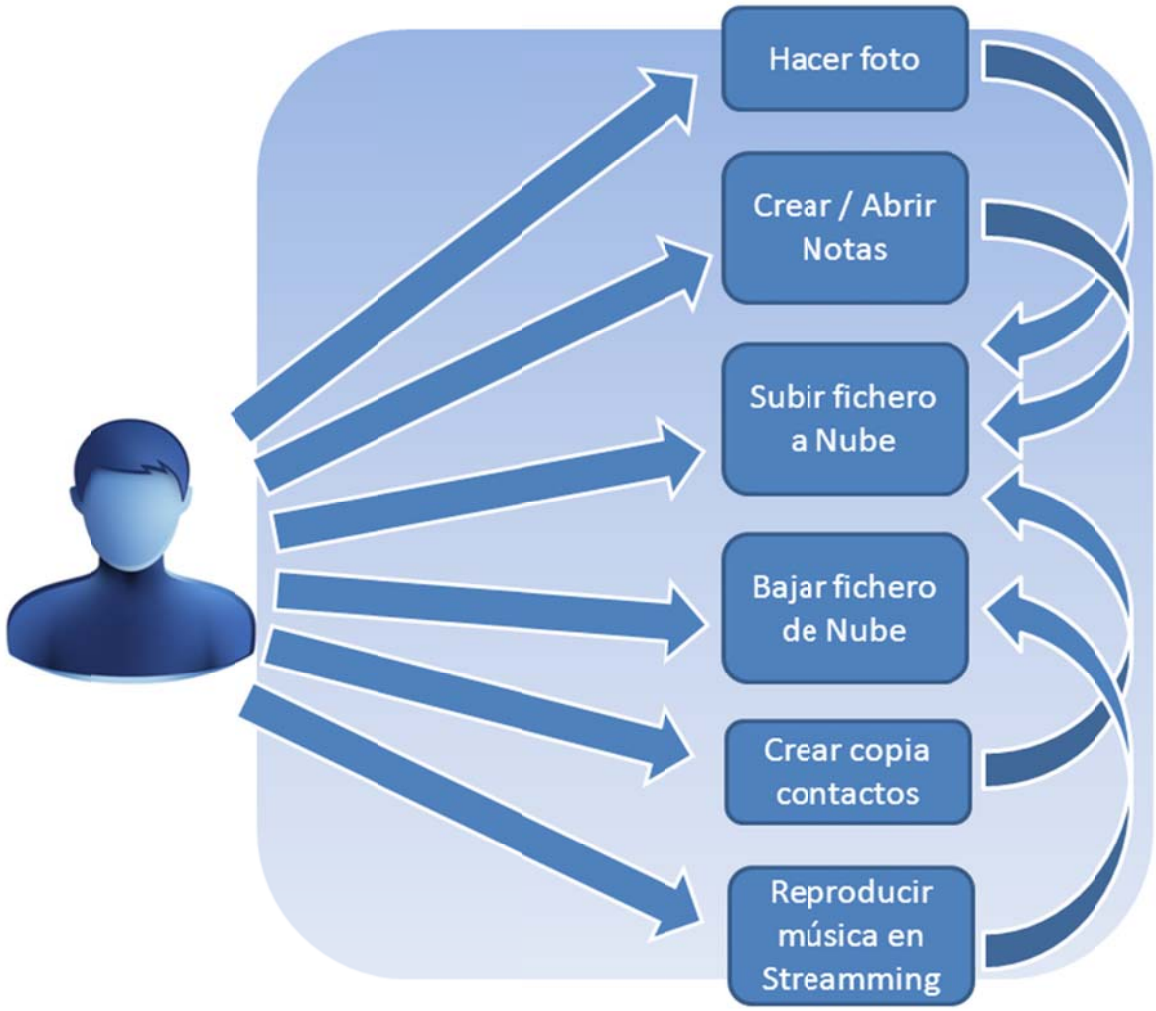
Arquitectura cliente de Copy2Cloud se divide en según el siguiente modelo de capas.

- La capa de presentación formada por los recursos textuales y la interfaz gráfica.
- Los permisos que permiten a la aplicación acceder a diferentes características del dispositivo.
- La lógica de aplicación formada por las diferentes clases que dan forma a la aplicación.

Esta separación de capas es la habitual en las aplicaciones para Android.

8.2. Diagramas de clases y usos implementados.

El diagrama de usos que se realiza en la entrega final es el siguiente.

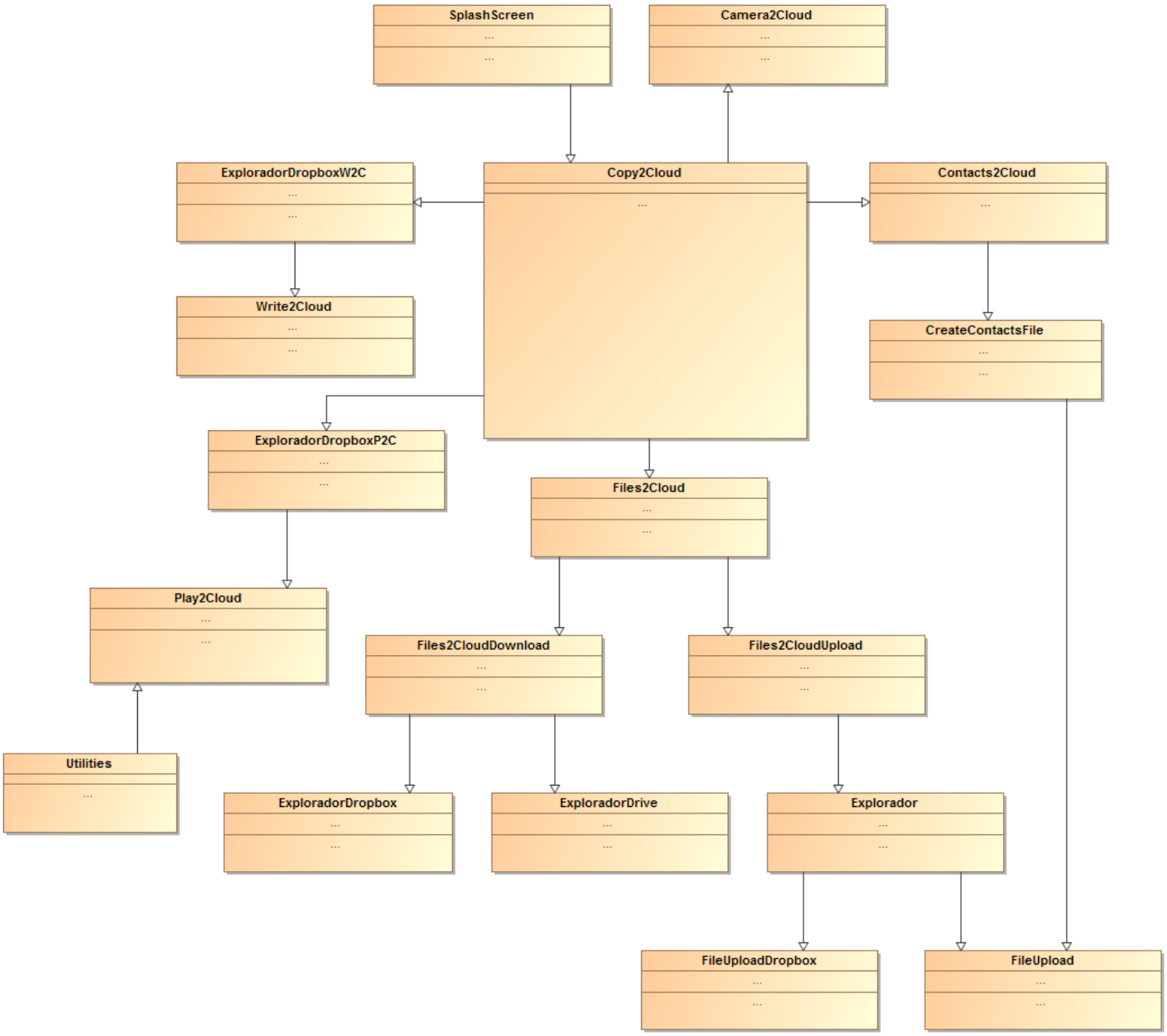


El usuario tiene una serie de opciones básicas a realizar:

- Hacer una foto y subirla al servicio de nube.
- Crear notas nuevas y abrir notas.
- Subir fichero a servicio de nube.
- Bajar fichero desde servicio de nube.
- Crear copia de contactos en nube.
- Reproducir música en streamming.

Todas estas tareas se apoyan en dos principales, subir fichero del servicio de nube y bajar fichero del servicio de nube, que realizan el intercambio entre la implementación de las APIs y el servicio de red.

El proyecto presenta el siguiente diagrama de clases.

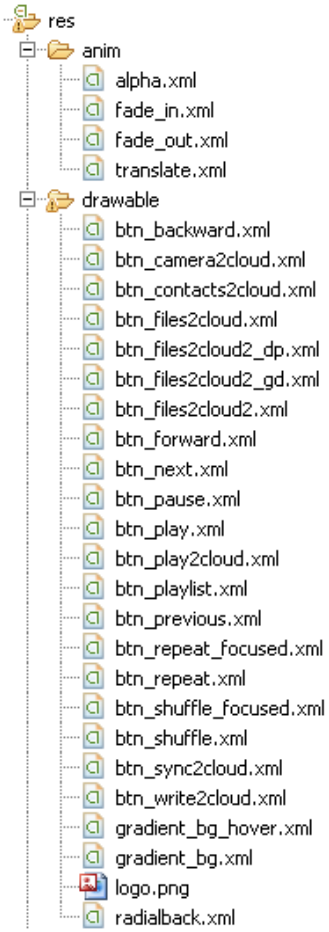


La aplicación comienza con una breve presentación del logo de Copy2Cloud apoyándose en la clase SplashScreen.java, y pasa al panel principal de la aplicación Copy2Cloud.java, lo que es el núcleo desde donde accederemos a todas las actividades que podremos realizar.

El resto de clases parten de la clase principal, y de estas a sus propias subclases sucesivas hasta completar la tarea.

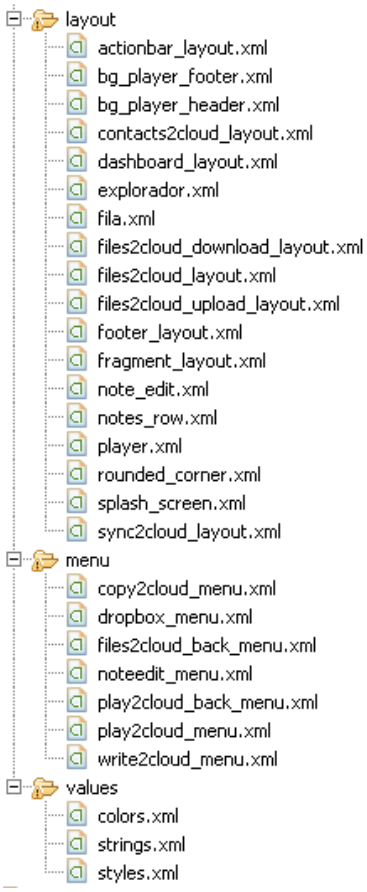
8.3. Capa de presentación

La capa de presentación se encuentra en el directorio *Copy2Cloud\res* de la aplicación.



Dentro del directorio de resources del proyecto, tenemos en la carpeta anim, las animaciones en las que se apoya la clase SplashScreen.java, que se utiliza para presentar el logo de Copy2Coud con un efecto de *"fade in"* con una elevación y traslado de éste y un posterior efecto de *"fade out"*, antes de presentar el panel principal.

La carpetas de drawable y drawable-hdpi contienen una gran cantidad de recursos de imágenes y ficheros xml en el que se encuentran, por ejemplo, los fragmentos en los que se compone el panel principal, las animaciones de los botones de los paneles con tres estados (default, selected y pressed), elementos para la composición del panel del reproductor de música, los logos de publicación de la aplicación, etc.



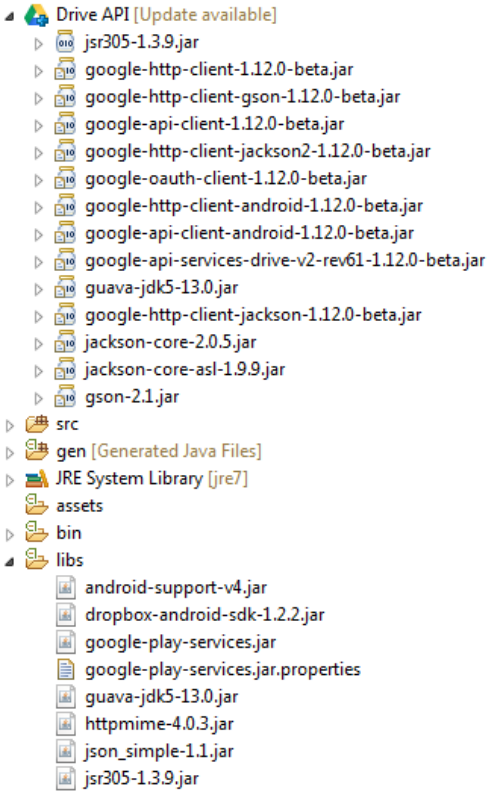
La carpeta layout, contiene los diseños de todos los paneles y fragmentos de la aplicación, estos ficheros xml los llamamos desde las clases para presentar visualmente cada pantalla.

Cada layout posee una opción de menú, que se guarda en la carpeta de recursos menu, para complementar y ayudar en la navegabilidad. Por ejemplo, el menú *noteedit_menu.xml* posee tres botones uno para salvar la nota, otro para abrir una y un último para cerrar. La mayoría de ellos son simples, y presentan una flecha de opción de regreso.

Y por último en la carpeta values, tenemos tres ficheros xml, uno de colores definidos, otro con tras las cadenas y mensajes utilizados en los layouts y los estilos definidos.

8.4. Carpetas de librerías y clases de la aplicación

Como sabemos, las clases manejan la capa de la lógica de la aplicación y éstas se apoyan en las librerías que nos facilitan el acceso a funciones preestablecidas que realizarán todas las acciones que necesitamos para que funcione correctamente nuestra aplicación.

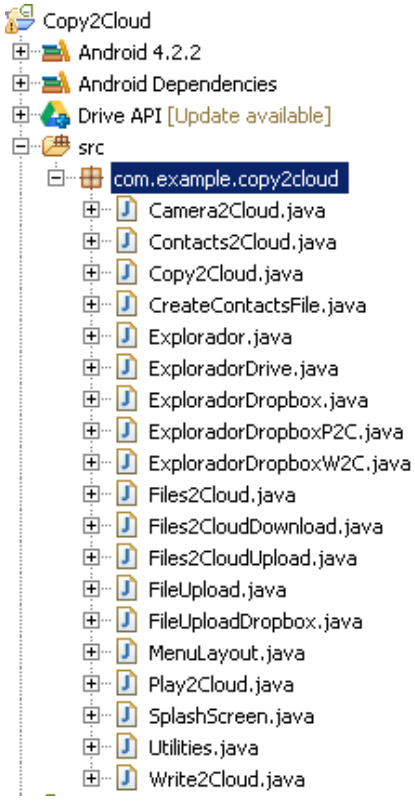


Las librerías se encuentran dentro de la carpeta libs y en la carpeta Drive API de nuestro proyecto.

Destacamos aquellas que necesitamos que son necesarias para la SDK de Google drive y las de la SDK de Dropbox. En el siguiente apartado explicaremos como obtenerlas y configurarlas en nuestro proyecto.

El resto de librerías realizan tareas de apoyo para los procesos de intercambios entre los servicios de ficheros y el cliente en el dispositivo Andriod.

También es necesaria la librería de soporte android-suport-v4.jar la cual tendremos que referenciar en el path de librerías del proyecto.



La carpeta src, dentro del paquete com.example.copy2cloud tenemos todas las clases que controlan la lógica de la aplicación Copy2Cloud.

Durante todo el proyecto se ha ido incrementando y variando el listado de actividades y clases de apoyo, tratando de dar versatilidad a la aplicación y utilizando los métodos de pruebas y fallos.

8.5. Permisos necesarios - el fichero AndroidManifest.xml

Este fichero en formato XML, incluye los datos relativos a las activities (pantallas) que forman la aplicación y las versiones de SDK para los que se desarrolla la misma, si no las añadimos en éste fichero de configuración, no funcionarán en el momento de la ejecución, con lo cual hay que tener muy claro que clases son las que se visualizan.

Los permisos usados en la aplicación son los referentes al acceso a cuentas de usuario, acceso a internet, acceso a los contactos telefónicos, acceso de escritura al almacenamiento externo y el manejo de las cuentas de usuario del dispositivo.

```
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.MANAGE_ACCOUNTS"/>
```

Además para el acceso de la API de Dropox necesitamos lo siguiente:

```
<activity
    android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar"
    android:name="com.example.copy2cloud.SplashScreen" >
    <intent-filter >
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity
    android:name="com.dropbox.client2.android.AuthActivity"
    android:configChanges="orientation|keyboard"
    android:launchMode="singleTask" >
    <intent-filter>
        <data android:scheme="[redacted]" />
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.BROWSABLE" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
```

En este se especifica la APP Key que genera el proceso de registro en la consola de Dropbox que explicaremos en un próximo apartado.

Las clases de tipo Activity siguen el ciclo de vida de una aplicación Android.

1. **Estado de inicio (Starting):**

Cuando una actividad todavía no existe en la memoria, podemos decir que es el estado de partida.

2. **Estado de ejecución (Running):**

Una actividad que está en primer plano está en el estado de ejecución. Cualquier actividad que se encuentra actualmente en la pantalla y permite la interacción con el usuario, esta actividad se ejecuta en ese punto de tiempo, se considera que está el plano superior de la pila de Actividades.

3. Estado pausado (Paused):

Cuando una actividad no está en el foco principal (es decir, no interactúa con el usuario), pero todavía es visible en la pantalla, que se encuentra en el estado de pausa.

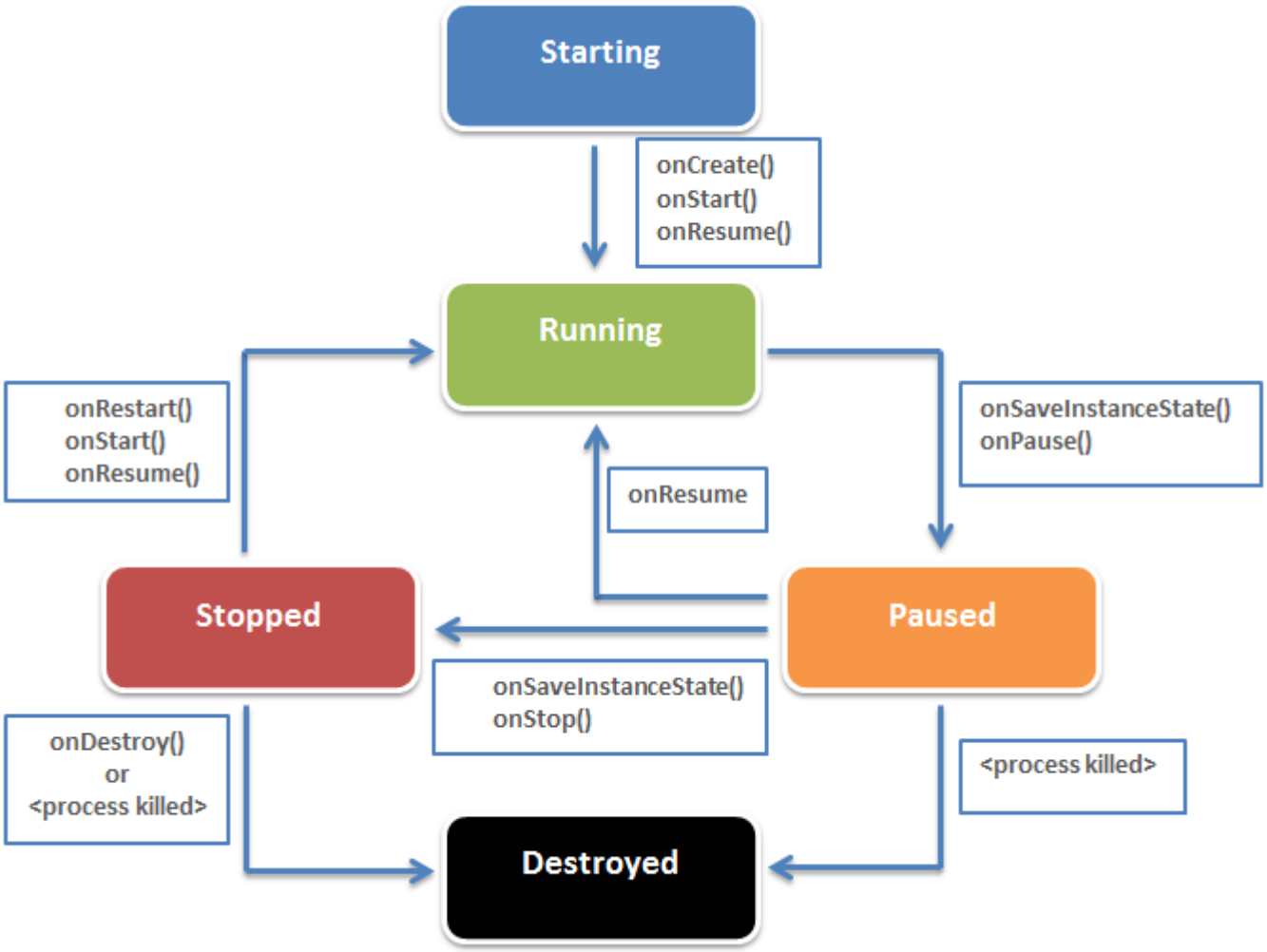
4. Estado Detenido (Stopped):

Una actividad que no es visible en la pantalla, pero existe en la memoria está en el estado detenido.

5. Estado Destruido (Destroyed):

Este estado, es el resultado de la eliminación de una actividad (que ya no es necesaria) de la memoria principal. Estos movimentos ocurren generalmente en un dispositivo Android, cuando el gestor de actividades decide que esta activida no va a ser usada más.

El siguiente esquema resume lo explicado.



8.6. Características de las APIs de para el desarrollo de la aplicación

Aunque inicialmente la implementación de la aplicación iba a ser con un servicio de nube solamente, inclinándome por **Google Drive**, y siendo el desarrollo de esta para el sistema Andriod, el estudio y la práctica real con éste me ha conducido a realizarlo con otra API del mismo tipo, decantándome por otro servicio de mayor aceptación como es **Dropbox**.

Comencemos comparando que ofrecen cada uno de los servicios y las características básicas de estos.

Características	Google Drive	Dropbox
Capacidad Almacenamiento gratuita	5GB	2GB
Máxima Capacidad Almacenamiento	16TB	100GB
Sincronización	Si	Si
Capacidades Offline	Si	Si
Compartir carpetas	Si	Si
Mobile Apps	Android (iPhone, iPad)	iPhone, iPad, BlackBerry, Android
Encriptación de datos	Si	SSL and AES-256 bit
Edición Simultanea	Si	No
Control de cambios	Si	Si
Tamaño Máximo	10GB	2GB

A primera vista, como se puede observar con claridad, Google ofrece mejor cartera de servicios y mayores capacidades, lo cual hace pensar que la implementación para desarrolladores también es mucho más ambiciosa.

Explicaremos gráficamente como funcionan ambos servicios de cara a la implementaci

8.7. Configuraciones de las consolas de aplicaciones de los servicios

Para el desarrollo de la aplicación, ha sido necesario el registro de ésta en las consolas de aplicaciones de los respectivos servicios de fichero en nube. De esta manera nos permite el acceso a los servicios implementados en la aplicación, ambas necesitan para el registro la ruta de la aplicación (com.example.copy2cloud). Procedemos a mostrar brevemente los procesos básicos de registro y configuración de éstos en el entorno de desarrollo en Eclipse.

1. Consola de aplicación de Google

Comenzaremos con la consola de aplicación de Google, si disponemos de un usuario de correo de Gmail será suficiente para autenticarnos en el servicio de APIs de Google. Para acceder al servicio lo hacemos en la siguiente dirección:

https://code.google.com/apis/console

Ahora antes de Crear nuestra ID y registrar la aplicación realizamos lo siguiente.

Paso 1: Generar la huella digital de firma de certificado (SHA1)

Google verifica las solicitudes de la API de Drive, que son enviados desde dispositivos Android, siendo necesario el nombre del empaquetado de la aplicación y una huella digital generada con SHA1 del certificado con firma con los definidos por la aplicación. Para ello necesitaremos generar nuestro propio certificado con el que codificaremos la aplicación al publicar la apk, para ello usaremos el siguiente comando **keytool** desde una ventana de terminal:

```
keytool -exportcert -alias androiddebugkey -keystore ~/.android/debug.keystore -list -v
```

Entonces nos pedirá, aparte de datos personales, la introducción de una contraseña para el almacén de claves, keytool posteriormente imprime por pantalla la huella digital (fingerprint), como por ejemplo:

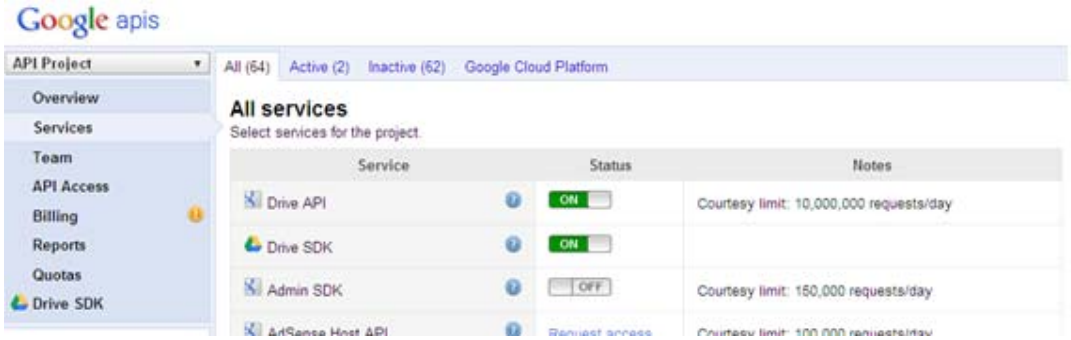
```
Certificate fingerprints:
  SHA1: 21:45:BD:F6:98:B8:71:50:39:BD:0E:83:F2:06:9B:ED:43:5A:C2:1C
```

Conservaremos esta cadena con la huella digital a mano para proceder a realizar el siguiente paso, activar la API de Drive.

Paso 2: Habilitar la API de Drive

Necesitaremos habilitar el API Drive para poder utilizarla en nuestra aplicación. Para realizar esto accederemos a la consola de aplicaciones en la dirección dada y seguimos los siguientes pasos:

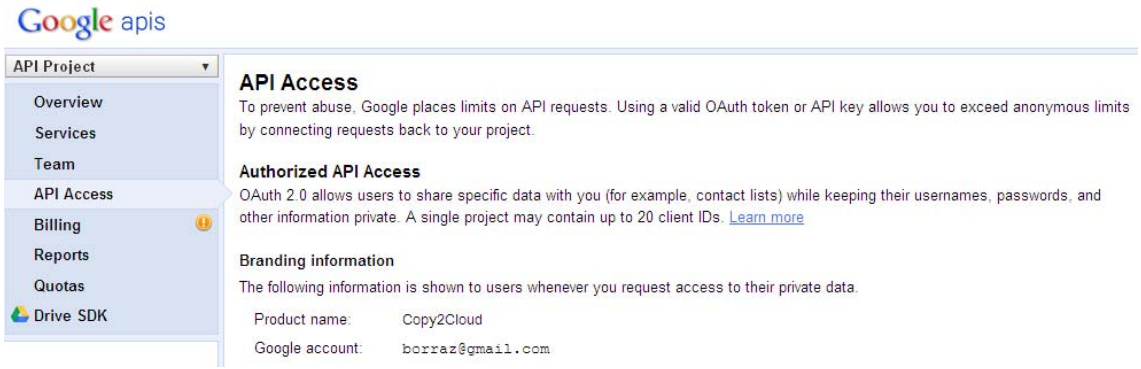
1. Creamos un proyecto de API en la consola de APIs de Google.
2. Seleccionamos la ficha Servicios (*) en el proyecto de API, y activar la API y la SDK de Drive.



(*) Ficha de servicios de Google APIs.

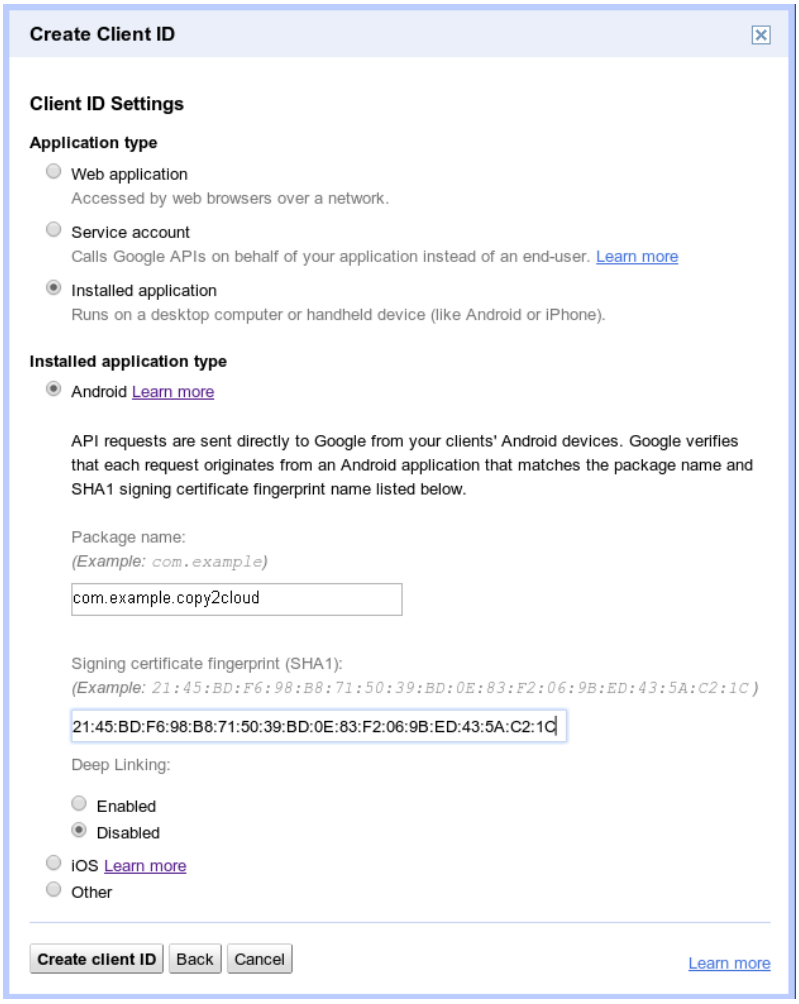
3. Ahora seleccionamos la opción de API Access para crear un Client ID OAuth 2.0.

4. En la opción de información de marca comercial, ponemos nuestro nombre de la aplicación “Copy2Cloud” y hacemos clic en Siguiente. Podemos añadir opcionalmente un logotipo y la URL de la página comercial.



5. Ahora en la sección Configuración del Client ID, hacemos lo siguiente:

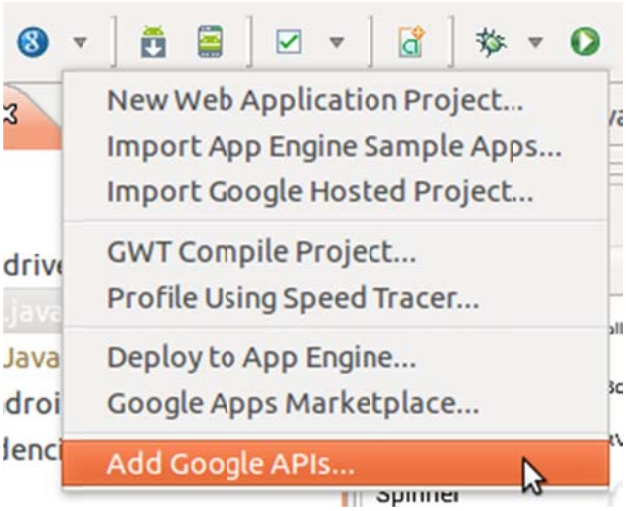
   a. Seleccionamos la opción de Installed application.
   b. Seleccionamos Android para el tipo de aplicaciones instaladas.
   c. Añadimos el nombre del paquete de la aplicación (com.example.copy2cloud).
   d. Pegamos la huella digital que previamente habíamos generado con keytool en el paso 1.
   e. Y por último, clicamos en Crear Client ID.



Con esto ya tendríamos autorizada nuestra aplicación para el acceso y uso de la SDK y API de Google Drive. Pasaremos al último paso para añadir las librerías en eclipse y seleccionar las configuraciones necesarias para que funcione correctamente.

Paso 3: Crear y configurar la SDK en el proyecto Android

a. En Eclipse, creamos un nuevo proyecto de aplicación para Android, con el nombre especificado en la consola.
b. Seleccionamos la API 17: Android y la 4.2 para el SDK de destino.
c. Seleccione las API de Google (Google Inc.) (17 API) para la compilación con.
d. Dejamos el resto de los campos con los valores predeterminados y finalizamos el asistente.
e. El siguiente paso es agregar la biblioteca de cliente de Google Drive para el proyecto, con el complemento de Google para Eclipse es fácil realizar esta tarea:
    1. Seleccionamos Añadir Google API en el complemento de Google para Eclipse.
    2. Seleccionamos Drive API v2 de la lista y hacemos clic en Finalizar.



Por último, es necesario añadir que la librería de **Google Play Services** esté incluido en el proyecto, para esto hacemos lo siguiente:

1. Seleccionamos *Proyect > Propierties > Java Build Path > Libraries* en el menú de Eclipse.
2. Añadimos el fichero **google-play-services.jar** en añadir JAR externos, el cual buscaremos en la carpeta de SDK de Android*/extras/google/google_play_services / libproject/google-play-services_lib/libs.*
3. Por último en la pestaña *Order and Export*, nos aseguramos de que la casilla de verificación por el archivo google-play-services.jar está marcada.

Y con esto ya tenemos preparado el entorno podemos empezar a trabajar con la API de Drive.

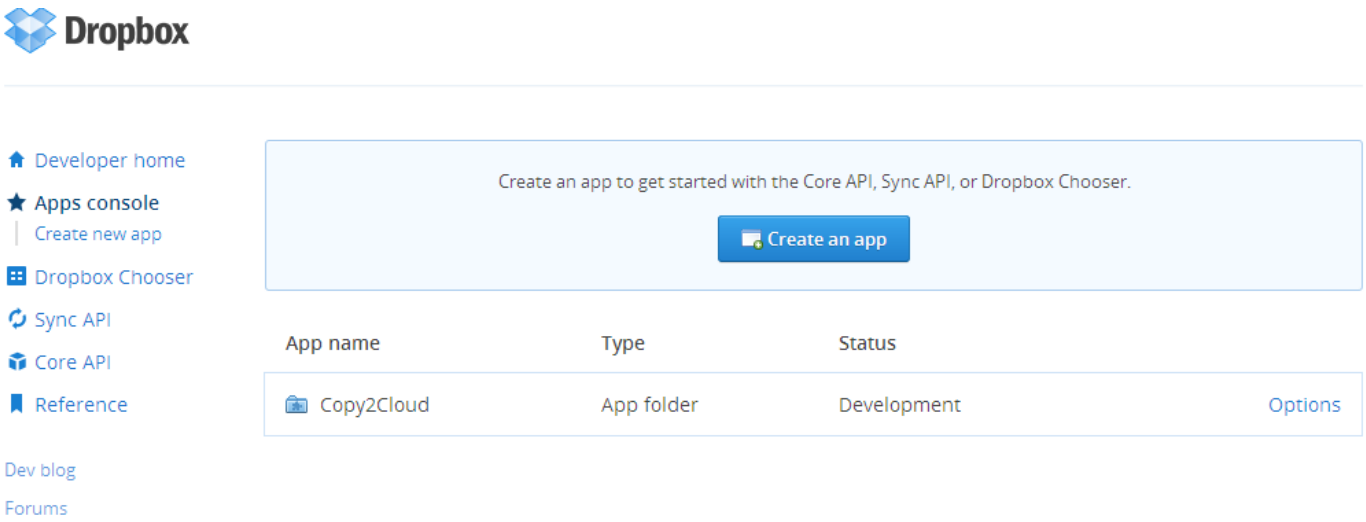
2. Consola de aplicación de Dropbox

Proseguimos con la consola de aplicación de Dropbox, para ello es suficiente estar registrado en el servicio y acceder con nuestro usuario con el que accedemos al servicio de ficheros, para acceder al servicio lo hacemos en la siguiente dirección:

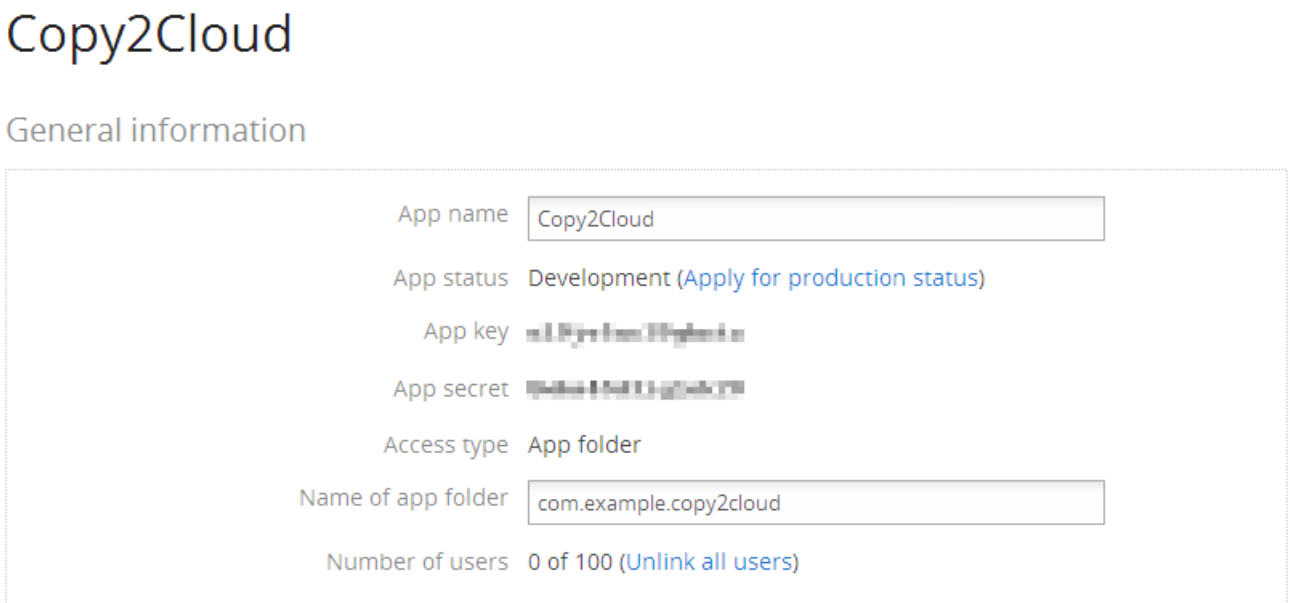
https://www.dropbox.com/developers

Paso 1: Habilitar la API de Dropbox

En este punto, al igual que la consola de Google, antes de crear y registrar nuestra aplicación.



Y al igual que con la consola de Google, añadimos el nombre de la ruta de nuestra aplicación y nos generará una **App key** y una **App secret**, las cuales usaremos en el código para poder autenticar nuestra aplicación y autorizar todos movimiento que hagamos con ésta.



Como podemos comprobar, el registro en la consola es mucho más sencillo. Ahora procederemos a cargar la SDK en nuestro proyecto.

Paso 2: Crear y configurar la SDK en el proyecto Android

Descargamos y descomprimimos la última versión de la SDK de Dropbox (dropbox-sync-sdk-android.jar) para Android, que se encuentra en la siguiente ruta:

https://dl.dropboxusercontent.com/sh/j6rb54nt2erea90/lOoNSs15sa/libs/dropbox-sync-sdk-android.jar?token_hash=AAF61XEyBellQ9zZb6imxqSFeBzw44muBJ7xLQA7qVFRMA&dl=1

Tendremos que incluir el fichero JAR de la carpeta *lib* en la ruta de nuestro proyecto, y con esto ya podremos utilizar la API de Dropbox, de nuevo se confirma la facilidad de configuración de este servicio para desarrolladores.

8.8. Desarrollo de las interfaces

Hemos tomando como base, todo lo desarrollado en la fase de diseño centrado a usuario, es decir los sketches y los prototipos, se han realizado las diferentes interfaces de usuario en los diferentes layouts utilizados, el listado de éstos son:

- **dashboard_layout.xml**
  - actionbar_layout.xml
  - fragment_layout.xml
  - footer_layout.xml
- **contacts2cloud_layout.xml**
- **files2cloud_layout.xml**
- **sync2cloud_layout.xml**
- **explorador.xml**
  - fila.xml
- **files2cloud_upload_layout.xml**
- **files2cloud_download_layout.xml**
- **player.xml**
  - bg_player_footer.xml
  - bg_player_header.xml
  - rounded_corner.xml
- **note_edit.xml**
  - notes_row.xml
- **splash_screen.xml**

Algunos son una composición de otros layouts, como es el caso de dashborard_layout.xml que se compone de un encabezado, un fragmento y un pie.

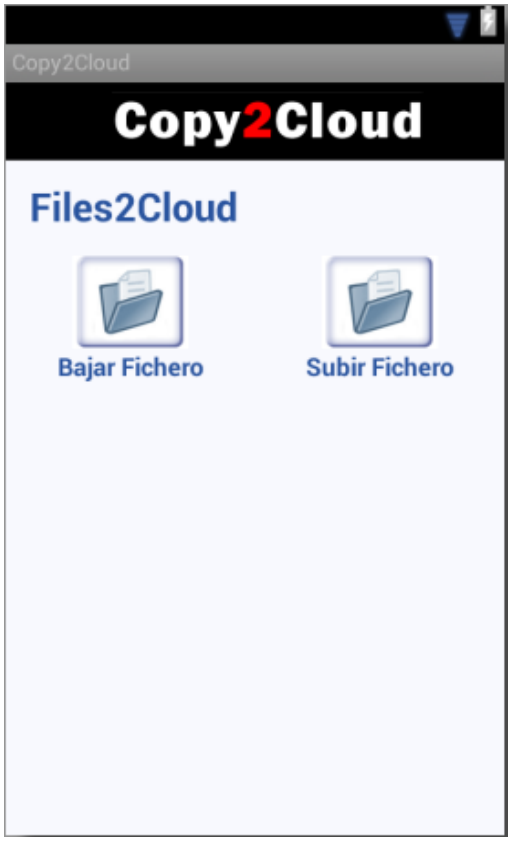
El trabajo de diseño es un trabajo laborioso, la verdad que aunque la ADT de eclipse da las herramientas básicas para el desempeño de este, la composición se hace laboriosa.

Proseguiremos a continuación, mostrando las interfaces finales en el dispositivo:

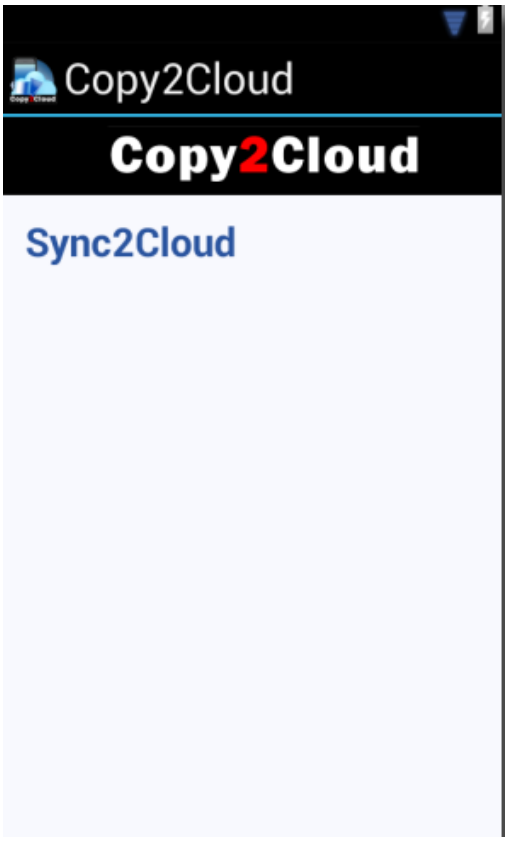
- **dashboard_layout.xml (actionbar_layout.xml, fragment_layout.xml y footer_layout.xml)**

- **contacts2cloud_layout.xml**

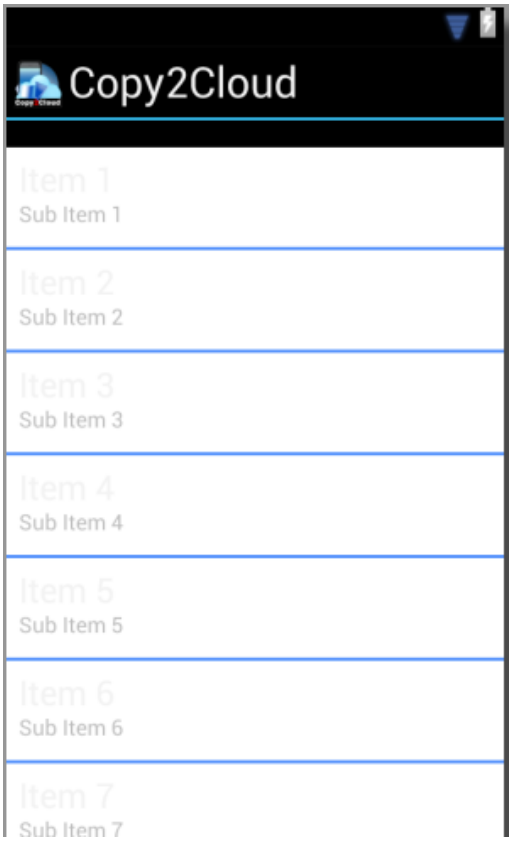
- files2cloud_layout.xml



- sync2cloud_layout.xml



- explorador.xml (incluye fila.xml)



- files2cloud_upload_layout.xml

- files2cloud_download_layout.xml

- player.xml (incluye bg_player_footer.xml, bg_player_header.xml y rounded_corner.xml)

- note_edit.xml (incluye notes_row.xml)

- splash_screen.xml

8.9. Programación de los módulos y funcionamiento de las clases

Se han desarrollado en 19 clases, las explicamos a continuación:

- **com.example.copy2cloud.Camera2Cloud.java**.

Esta clase realiza una foto y la sube al servicio de nube Google Drive. Primero permite seleccionar la cuenta de usuario de Google, ya introducida el dispositivo Android o crear una nueva, presenta las credenciales a Google Drive y en un *Intent* conecta con el dispositivo de la cámara fotográfica.

Una vez realizada la foto, la formatea con la fecha realizada y procede a subirla a Google Drive.

Termina mostrando por pantalla un *Toast* anunciando la subida exitosa de la foto al servicio y vuelve a pedir las credenciales.

- **com.example.copy2cloud.MenuLayout.java**

Esta clase realiza un diseño personalizado que dispone los iconos del panel principal de la aplicación en forma de rejilla, realiza una optimización de la distribución de los espacios en blanco de la pantalla, cuando se visualiza en formato horizontal y en formato vertical.

- **com.example.copy2cloud.Contacts2Cloud.java**

Esta clase presenta un panel sencillo en el que se ubica el título del panel Conctacs2Cloud con un botón que realizará la copia de seguridad de los contactos, al pulsarse el botón ejecutará la clase *CreateContactFile.java*.

Presenta también un menú de opciones al pulsar el botón de menú que permite volver hacia atrás.

- **com.example.copy2cloud.Files2Cloud.java**

Esta clase presenta un panel sencillo en el que se ubica el título del panel Files2Cloud con dos botones uno para *Bajar Fichero* y otro para *Subir Fichero*, éstos conducen a las dos clases *Files2CloudUpload.java* y *Files2CloudDownload.java*.

Presenta también un menú de opciones al pulsar el botón de menú que permite volver hacia atrás.

- **com.example.copy2cloud.Copy2Cloud.java**

Esta clase contiene el panel principal de la aplicación, con una botonera de 6 botones que conducen a cada una de las tareas de una forma clara y contundente.

Presenta también un menú de opciones al pulsar el botón de menú que permite cerrar la aplicación, muestra un menú de información con un botón de aceptación.

- **com.example.copy2cloud.Explorador.java**

Esta clase realiza un listado de las carpetas y ficheros en el almacenamiento exterior del dispositivo móvil, separa por un lado las carpetas y por otro lado los ficheros de forma ordenada, y permite la navegación desde la raíz de la tarjeta de memoria */sdcard/*.

Tiene implementado un método onListItemClick() que devuelve la cadena del path del fichero que se va a subir al servicio de nube.

Presenta también un menú de opciones al pulsar el botón de menú que permite volver hacia atrás.

- **com.example.copy2cloud.CreateContactsFile.java**

Esta clase realiza la copia de los contactos del dispositivo móvil en un fichero VCF (vCard) formateado con la fecha de realización, lo guarda en el raíz de la */sdcard* y llama a la clase *UploadFile.java* para subirlo al servicio de nube Google Drive.

- **com.example.copy2cloud.Files2CloudUpload.java**

Esta clase presenta un panel sencillo en el que se ubica el título del panel Files2Cloud – Upload con tres botones uno para seleccionar el fichero a subir, el cual llama a la clase *Explorador.java* y otros dos para subir el fichero a Dropbox o a Google Drive, si no se selecciona el fichero con el explorador no permitirá subir los ficheros mostrando un mensaje en un *Toast*, una vez seleccionado el botón de subir a Dropbox llamará a la clase *UploadFileDropbox.java* y el botón de subir fichero a Google Drive llamará a la clase *UploadFile.java*.

Presenta también un menú de opciones al pulsar el botón de menú que permite volver hacia atrás.

- **com.example.copy2cloud.Files2CloudDownload.java**

Esta clase presenta un panel sencillo en el que se ubica el título del panel Files2Cloud – Download con dos botones uno para seleccionar el fichero para visualizar únicamente un listado de ficheros de Google Drive, que conecta con la clase ExploradorDrive.java y otro para descargar ficheros desde Dropbox que conecta con la clase ExploradorDropbox.java.

Presenta también un menú de opciones al pulsar el botón de menú que permite volver hacia atrás.

- **com.example.copy2cloud.FileUpload.java**

Esta clase no se apoya en ningún layout, realiza la acción de subir un fichero a Google Drive, pide las credenciales y autoriza el servicio, una vez subido el fichero muestra un *Toast* con la información del proceso realizado de forma exitosa.

- **com.example.copy2cloud.Utilities.java**

Esta clase sirve de apoyo al reproductor de música de la clase *Play2Cloud.java*. Básicamente realiza las funciones que calculan y convierten el tiempo de milisegundos a un formato de tiempo real para el reproductor, devolviéndolo en una String, también calcula el porcentaje de progreso y la duración de la canción.

- **com.example.copy2cloud.Play2Cloud.java**

Esta clase es un reproductor multimedia, con una barra de progreso, acceso al listado de conaciones en Dropbox, y el juego de botones de reproducción, pausa, etc.

Presenta un botón que en el layout se ve en la parte superior derecha que llama a la clase *ExploradorDropboxP2C.java* que muestra las canciones subidas al servicio de nube.

Presenta también un menú de opciones al pulsar el botón de menú que permite volver al menú principal mientras se reproduce la música en segundo plano.

- **com.example.copy2cloud.ExploradorDrive.java**

Esta clase realiza un listado de los ficheros en Google Drive, primeramente vuelve a solicitar la cuenta de usuario con la que se va a realizar la tarea y entonces muestra el listado. Como no me ha sido posible implementar la descarga autorizada de este servicio, solamente muestra dicho listado pero al hacer click no hace nada.

Presenta también un menú de opciones al pulsar el botón de menú que permite volver hacia atrás.

- **com.example.copy2cloud.FileUploadDropbox.java**

Esta clase no se apoya en ningún layout, realiza la acción de subir un fichero a Dropbox procedente de las clases *ExploradorDropbox.java*, *ExploradorDropboxP2C.java* y *ExploradorDropbox.javaW2C*. Ésta requiere las credenciales de Dropbox a través del navegador y solicita autorizar el servicio por parte del usuario, una vez hecho esto ya no volverá a pedir el usuario y contraseña.

Una vez subido el fichero muestra un *Toast* con la información del proceso realizado de forma exitosa.

- **com.example.copy2cloud.ExploradorDropbox.java**

Esta clase realiza un navegador de ficheros de red, permite al usuario moverse por los ficheros de Dropbox como si fuera un sistema de ficheros local, y cuando se selecciona un fichero te pregunta si éste quiere ser descargado o no.

Presenta también un menú de opciones al pulsar el botón de menú que permite volver hacia atrás.

- **com.example.copy2cloud.ExploradorDropboxP2C.java**

Esta clase realiza un navegador de ficheros de red de la carpeta de ficheros mp3 que reproducirá la clase *Play2Cloud.java*, permite al usuario moverse por ésta como si fuera un sistema de ficheros local, y cuando se selecciona un fichero te pregunta si éste quiere ser reproducida la canción o no.

Presenta también un menú de opciones que conecta con la clase *Files2CloudUpload.java* para subir ficheros mp3 a la carpeta de Dropbox. Para realizar esto correctamente, primero debemos crear una carpeta en la raíz de */sdcard/P2C* y poner ahí los ficheros para subirlos.

- **com.example.copy2cloud.SplashScreen.java**

Esta clase realiza la animación de inicio de la aplicación, manejando el layout con el logo de Copy2Cloud, y usando los ficheros XML de la carpeta de recursos *anim*. Posteriormente conecta con la clase principal *Copy2Cloud.java* que nos mostrará el panel principal.

- **com.example.copy2cloud.Write2Cloud.java**

Esta clase es básicamente un editor de notas, tiene dos campos EditText y uno TextView, los dos primeros como sabemos editables y y el otro muestra simplemente la fecha actual. En el EditText superior se ubica el título y en el inferior (el cuerpo de la nota) se introduce el texto a anotar.

Presenta también un menú de opciones al pulsar el botón de menú que muestra tres opciones, salvar la nota editada, abrir otra nota (este botón conecta con la clase *ExploradorDropboxW2C.java*) y cerrar la ventana de anotaciones.

- **com.example.copy2cloud.ExploradorDropboxW2C.java**

Esta clase realiza un navegador de ficheros de red de la carpeta de ficheros TXT que abrirá la clase *Write2Cloud.java*, permite al usuario moverse por ésta como si fuera un sistema de ficheros local, y cuando se selecciona un fichero te pregunta si éste quiere ser mostrado o no.

Presenta también un menú de opciones que permite crear una nota en blanco.

8.10. Código fuente de las clases de Copy2Cloud

A continuación se expone todo el código fuente completo utilizado para todas las clases de la aplicación Copy2Cloud.

Comentar que las APP Keys y APP Secret utilizadas para validar el servicio de la SDK de Dropbox las he ocultado ya que las considero personales y si se desea implementar se deberán general tal y como se explica en el apartado 8.7 de esta misma memoria.

El resto del código lo considero de libre difusión.

1. Contacts2Cloud.java

```java
package com.example.copy2cloud;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;

import com.example.copy2cloud.R;

public class Contacts2Cloud extends Activity {
    // Called when the activity is first created.
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.contacts2cloud_layout);

        Button btn_files2cloud = (Button) findViewById(R.id.btn_conctats2cloud);

        btn_files2cloud.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View view) {
            // Launching Create Contacts File class
            Intent i = new Intent(getApplicationContext(), CreateContactsFile.class);
            startActivity(i);
        }
        });
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        try{
            finish();

            return super.onOptionsItemSelected(item);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
        }

        public boolean onCreateOptionsMenu(Menu menu) {
            // Inflate the menu; this adds items to the action bar if it is present.
            getMenuInflater().inflate(R.menu.play2cloud_back_menu, menu);
            return true;
  }
}

```

2. Copy2Cloud.java

```java
package com.example.copy2cloud;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import com.example.copy2cloud.R;

public class Copy2Cloud extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.dashboard_layout);

        /**
         * It makes all button instances
         * */
        // Dashboard Camera2Cloud button - Camera2Cloud
        Button btn_camera2cloud = (Button) findViewById(R.id.btn_camera2cloud);

        // Dashboard Files2Cloud button
        Button btn_files2cloud = (Button) findViewById(R.id.btn_conctats2cloud);

        // Dashboard Write2Cloud button
        Button btn_write2cloud = (Button) findViewById(R.id.btn_write2cloud);

        // Dashboard Play2Cloud button
        Button btn_play2cloud = (Button) findViewById(R.id.btn_play2cloud);

        // Dashboard Contacts2Cloud button - Contacts2Cloud
        Button btn_contacts2cloud = (Button) findViewById(R.id.btn_contacts2cloud);

        // Dashboard Sync2Cloud button
        Button btn_sync2cloud = (Button) findViewById(R.id.btn_sync2cloud);

        /**
         * It handle all buttons
         * */

        // Listening to Camera2Cloud button click
        btn_camera2cloud.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                // Launching Camera2Cloud Screen
                Intent i = new Intent(getApplicationContext(),
                        Camera2Cloud.class);
                startActivity(i);
            }
        });

        // Listening Files2Cloud button click
        btn_files2cloud.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                // Launching Files2Cloud Screen
                Intent i = new Intent(getApplicationContext(),
                        Files2Cloud.class);
                startActivity(i);
            }
        });

```

```
72          // Listening Write2Cloud button click
73          btn_write2cloud.setOnClickListener(new View.OnClickListener() {
74
75              @Override
76              public void onClick(View view) {
77                  // Launching Write2Cloud Screen
78                  Intent i = new Intent(getApplicationContext(),
79                          ExploradorDropboxW2C.class);
80                  startActivity(i);
81              }
82          });
83
84          // Listening to Play2Cloud button click
85          btn_play2cloud.setOnClickListener(new View.OnClickListener() {
86
87              @Override
88              public void onClick(View view) {
89                  // Launching ExploradorDropboxP2C Screen
90                  Intent i = new Intent(getApplicationContext(),
91                          ExploradorDropboxP2C.class);
92                  startActivity(i);
93              }
94          });
95
96          // Listening to Contacts2Cloud button click
97          btn_contacts2cloud.setOnClickListener(new View.OnClickListener() {
98
99              @Override
100             public void onClick(View view) {
101                 // Launching Contacts2Cloud Screen
102                 Intent i = new Intent(getApplicationContext(),
103                         Contacts2Cloud.class);
104                 startActivity(i);
105             }
106         });
107
108         // Listening to Sync2Cloud button click
109         btn_sync2cloud.setOnClickListener(new View.OnClickListener() {
110
111             @Override
112             public void onClick(View view) {
113                 // Launching Sync2Cloud Screen
114
115                 Toast.makeText(
116                         Copy2Cloud.this,
117                         "Opción no implementada para esta versión de Copy2Cloud.",
118                         Toast.LENGTH_LONG).show();
119
120             }
121         });
122     }
123
124     public boolean onOptionsItemSelected(MenuItem item) {
125         switch (item.getItemId()) {
126         case R.id.menu_cerrar:
127
128             AlertDialog.Builder dialog = new AlertDialog.Builder(
129                     Copy2Cloud.this);
130             dialog.setTitle("Cerrar Copy2Cloud");
131             dialog.setMessage("¡Gracias por usar Copy2Cloud!"
132                     + "\nDaniel Borraz Morón - TFC - Desarrollo de Aplicaciones para Dispositivos Móviles. "
133                     + "\ndanielborarz@uoc.edu");
134
135             dialog.setPositiveButton("OK",
136                     new DialogInterface.OnClickListener() {
137
138                         @Override
139                         public void onClick(DialogInterface dialog, int which) {
140                             dialog.cancel();
141                             finish();
142                         }
143                     });
144             dialog.show();
145             return true;
146
147         default:
148             return super.onOptionsItemSelected(item);
149         }
150
151     }
152
153     public boolean onCreateOptionsMenu(Menu menu) {
154         // Inflate the menu; this adds items to the action bar if it is present.
155         getMenuInflater().inflate(R.menu.copy2cloud_menu, menu);
156         return true;
157     }
158 }
```

3. Camera2Cloud.java

```java
package com.example.copy2cloud;


import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

import android.accounts.AccountManager;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.provider.MediaStore;
import android.widget.Toast;

import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.googleapis.extensions.android.gms.auth.GoogleAccountCredential;
import com.google.api.client.googleapis.extensions.android.gms.auth.UserRecoverableAuthIOException;
import com.google.api.client.http.FileContent;
import com.google.api.client.json.gson.GsonFactory;
import com.google.api.services.drive.Drive;
import com.google.api.services.drive.DriveScopes;
import com.google.api.services.drive.model.File;

public class Camera2Cloud extends Activity {
  static final int REQUEST_ACCOUNT_PICKER = 1;
  static final int REQUEST_AUTHORIZATION = 2;
  static final int CAPTURE_IMAGE = 3;

  private static Uri fileUri;
  private static Drive service;
  private GoogleAccountCredential credential;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    credential = GoogleAccountCredential.usingOAuth2(this, DriveScopes.DRIVE);
    startActivityForResult(credential.newChooseAccountIntent(), REQUEST_ACCOUNT_PICKER);
  }

  @Override
  protected void onActivityResult(final int requestCode, final int resultCode, final Intent data) {
    switch (requestCode) {
    case REQUEST_ACCOUNT_PICKER:
      if (resultCode == RESULT_OK && data != null && data.getExtras() != null) {
        String accountName = data.getStringExtra(AccountManager.KEY_ACCOUNT_NAME);

        if (accountName != null) {
          credential.setSelectedAccountName(accountName);
          service = getDriveService(credential);
          startCameraIntent();
        }
      }
      break;
    case REQUEST_AUTHORIZATION:
      if (resultCode == Activity.RESULT_OK) {
        saveFileToDrive();
      } else {
        startActivityForResult(credential.newChooseAccountIntent(), REQUEST_ACCOUNT_PICKER);
      }
      break;
    case CAPTURE_IMAGE:
      if (resultCode == Activity.RESULT_OK) {
        saveFileToDrive();
      }
    }
  }
```

```
71    //Call Photo Camera Intent
72    private void startCameraIntent() {
73      String mediaStorageDir = Environment.getExternalStoragePublicDirectory(
74          Environment.DIRECTORY_PICTURES).getPath();
75      String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss", Locale.US).format(new Date());
76      fileUri = Uri.fromFile(new java.io.File(mediaStorageDir + java.io.File.separator + "IMG_"
77          + timeStamp + ".jpg"));
78
79      Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
80      cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);
81      startActivityForResult(cameraIntent, CAPTURE_IMAGE);
82    }
83    //Save file to Drive
84    private void saveFileToDrive() {
85      Thread t = new Thread(new Runnable() {
86        @Override
87        public void run() {
88          try {
89            startActivityForResult(credential.newChooseAccountIntent(), REQUEST_ACCOUNT_PICKER);
90
91            // File's binary content
92            java.io.File fileContent = new java.io.File(fileUri.getPath());
93            FileContent mediaContent = new FileContent("image/jpeg", fileContent);
94
95            // File's metadata.
96            File body = new File();
97            body.setTitle(fileContent.getName());
98            body.setMimeType("image/jpeg");
99
100           File file = service.files().insert(body, mediaContent).execute();
101           if (file != null) {
102             showToast("Foto subida: " + file.getTitle());
103             startCameraIntent();
104           }
105         } catch (UserRecoverableAuthIOException e) {
106           startActivityForResult(e.getIntent(), REQUEST_AUTHORIZATION);
107         } catch (IOException e) {
108           e.printStackTrace();
109         }
110       }
111     });
112     t.start();
113   }
114   //Call Drive Service and build authorization
115   private Drive getDriveService(GoogleAccountCredential credential) {
116     return new Drive.Builder(AndroidHttp.newCompatibleTransport(), new GsonFactory(), credential)
117         .build();
118   }
119   //Show Toast
120   public void showToast(final String toast) {
121     runOnUiThread(new Runnable() {
122       @Override
123       public void run() {
124         Toast.makeText(getApplicationContext(), toast, Toast.LENGTH_LONG).show();
125       }
126         });
127     }
128 }
```

4. CreateContactsFile.java

```
package com.example.copy2cloud;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.res.AssetFileDescriptor;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.provider.ContactsContract;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
//It makes Contacts File in vcf format
public class CreateContactsFile extends Activity {
    Cursor cursor;
    ArrayList<String> vCard;

    final static String vfile = "Contacts" + "_" + System.currentTimeMillis()
            + ".vcf";
    static Context mContext;

    /** Called when the activity is first created. */
    @SuppressLint("SdCardPath")
    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        mContext = CreateContactsFile.this;
        getVCF();
        showToast("Fichero de Contactos Creado: " + vfile);
        //Passes path and name of vcf file to FileUpload.class to Upload File to Google Drive
        Intent i = new Intent(getApplicationContext(), FileUpload.class);

        i.putExtra("direccion", "/mnt/sdcard/" + vfile);
        startActivity(i);

        finish();
    }
    //It makes vcf File
    private static void getVCF() {

        Cursor phones = mContext.getContentResolver().query(
                ContactsContract.CommonDataKinds.Phone.CONTENT_URI, null, null,
                null, null);
        phones.moveToFirst();

        for (int i = 0; i < phones.getCount(); i++) {
            String lookupKey = phones.getString(phones
                    .getColumnIndex(ContactsContract.Contacts.LOOKUP_KEY));
            Uri uri = Uri.withAppendedPath(
                    ContactsContract.Contacts.CONTENT_VCARD_URI, lookupKey);
            AssetFileDescriptor fd;

            try {
                fd = mContext.getContentResolver().openAssetFileDescriptor(uri,
                        "r");
                FileInputStream fis = fd.createInputStream();
                byte[] buf = new byte[(int) fd.getDeclaredLength()];
                fis.read(buf);
                String VCard = new String(buf);
                String path = Environment.getExternalStorageDirectory()
                        .toString() + File.separator + vfile;
                @SuppressWarnings("resource")
```

```
72                  FileOutputStream mFileOutputStream = new FileOutputStream(path,
73                          true);
74                  mFileOutputStream.write(VCard.toString().getBytes());
75                  phones.moveToNext();
76
77              } catch (Exception e1) {
78
79                  e1.printStackTrace();
80              }
81          }
82      }
83
84      private void showToast(final String toast) {
85          runOnUiThread(new Runnable() {
86              @Override
87              public void run() {
88                  Toast.makeText(getApplicationContext(), toast,
89                          Toast.LENGTH_LONG).show();
90              }
91          });
92      }
93
94      public boolean onOptionsItemSelected(MenuItem item) {
95          try {
96              finish();
97
98              return super.onOptionsItemSelected(item);
99          } catch (Exception e) {
100             e.printStackTrace();
101         }
102         return false;
103     }
104
105     public boolean onCreateOptionsMenu(Menu menu) {
106         // Inflate the menu; this adds items to the action bar if it is present.
107         getMenuInflater().inflate(R.menu.play2cloud_back_menu, menu);
108         return true;
109     }
110 }
```

5. Explorador.java

```java
package com.example.copy2cloud;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.example.copy2cloud.R;

import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.Intent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
//Simple File SD Card Explorer
public class Explorador extends ListActivity {

    private List<String> item = null;
    private List<String> path = null;
    private String root;
    private TextView myPath;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.explorador);
        myPath = (TextView) findViewById(R.id.path);

        root = Environment.getExternalStorageDirectory().getPath();

        getDir(root);
    }
    //Get Dir Items List and build path
    private void getDir(String dirPath) {
        myPath.setText("Location: " + dirPath);
        item = new ArrayList<String>();
        path = new ArrayList<String>();
        File f = new File(dirPath);
        File[] files = f.listFiles();

        if (!dirPath.equals(root)) {
            item.add(root);
            path.add(root);
            item.add("../");
            path.add(f.getParent());
        }

        Arrays.sort(files);
        File file;

        for (int i = 0; i < files.length; i++) {
            file = files[i];

            if (!file.isHidden() && file.canRead()) {
                if (file.isDirectory()) {
                    path.add(file.getPath());

                    item.add(file.getName() + "/");
                }
            }
        }

        for (int i = 0; i < files.length; i++) {
            file = files[i];
```

```
72
73              if (!file.isHidden() && file.canRead()) {
74                  if (file.isFile()) {
75                      path.add(file.getPath());
76
77                      item.add(file.getName());
78                  }
79              }
80          }
81          ArrayAdapter<String> fileList = new ArrayAdapter<String>(this,
82                  R.layout.fila, item);
83          setListAdapter(fileList);
84      }
85      //On List Item Click saves selected Item to Files2CloudUpload Class
86      @Override
87      protected void onListItemClick(ListView l, View v, int position, long id) {
88          File file = new File(path.get(position));
89
90          if (file.isDirectory()) {
91              if (file.canRead()) {
92                  getDir(path.get(position));
93
94              } else {
95                  new AlertDialog.Builder(this)
96                          .setIcon(R.drawable.ic_launcher)
97                          .setTitle(
98                                  "[" + file.getName()
99                                          + "] carpeta inaccesible!!")
100                         .setPositiveButton("OK", null).show();
101             }
102         } else {
103             Intent data = new Intent();
104             data.setData(Uri.parse(path.get(position)));
105             setResult(RESULT_OK, data);
106             finish();
107         }
108     }
109
110     public boolean onOptionsItemSelected(MenuItem item) {
111         try {
112             finish();
113
114             return super.onOptionsItemSelected(item);
115         } catch (Exception e) {
116             e.printStackTrace();
117         }
118         return false;
119     }
120
121     public boolean onCreateOptionsMenu(Menu menu) {
122         // Inflate the menu; this adds items to the action bar if it is present.
123         getMenuInflater().inflate(R.menu.play2cloud_back_menu, menu);
124         return true;
125     }
126 }
```

6. ExploradorDrive.java

```java
package com.example.copy2cloud;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

import android.accounts.AccountManager;
import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.ListActivity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.text.method.ScrollingMovementMethod;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.googleapis.extensions.android.gms.auth.GoogleAccountCredential;
import com.google.api.client.googleapis.extensions.android.gms.auth.UserRecoverableAuthIOException;
import com.google.api.client.http.HttpTransport;
import com.google.api.client.json.gson.GsonFactory;
import com.google.api.services.drive.Drive;
import com.google.api.services.drive.DriveScopes;
import com.google.api.services.drive.model.File;
import com.google.api.services.drive.model.FileList;
//Google Drive File Explorer, It contents a List of all files (including recycle bin Files)
public class ExploradorDrive extends ListActivity {

    private final HttpTransport transport = AndroidHttp
            .newCompatibleTransport();
    private TextView output;
    private static Drive drive;

    private GoogleAccountCredential credential;
    private static final int CHOOSE_ACCOUNT = 1;
    private static final int REQUEST_AUTHORIZATION = 2;

    private List<String> item = null;
    private static FileList listf;
    private static String fdownloadUrl;
    private static String fname;
    // private static File file;
    private String token = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.explorador);

        output = (TextView) findViewById(R.id.path);
        output.setMovementMethod(new ScrollingMovementMethod());

        credential = GoogleAccountCredential.usingOAuth2(this,
                DriveScopes.DRIVE);
        startActivityForResult(credential.newChooseAccountIntent(),
                CHOOSE_ACCOUNT);
    }
```

```
73      @SuppressLint("NewApi")
74      @Override
75      public void onActivityResult(int requestCode, int resultCode, Intent data) {
76          switch (requestCode) {
77          case CHOOSE_ACCOUNT:
78              if (data != null) {
79                  Bundle extras = data.getExtras();
80                  if (extras != null) {
81                      String accountName = extras
82                              .getString(AccountManager.KEY_ACCOUNT_NAME);
83
84                      if (accountName != null) {
85                          credential.setSelectedAccountName(accountName);
86                          drive = new Drive.Builder(transport, new GsonFactory(),
87                                  credential).build(); // transport , jsonFactory
88                      }
89                  }
90              }
91
92              break;
93          case REQUEST_AUTHORIZATION:
94              if (resultCode == Activity.RESULT_OK) {
95                  Toast.makeText(this,
96                          "Clickea de nuevo el boton para realizar la acción.",
97                          Toast.LENGTH_LONG).show();
98
99              } else {
100                 startActivityForResult(credential.newChooseAccountIntent(),
101                         CHOOSE_ACCOUNT);
102
103             }
104             break;
105         }
106         listFilesDrive(null);
107     }
108
109     /**
110      * Downloads list of files from Drive account and presents them in TextView.
111      */
112     public void listFilesDrive(View v) {
113         output.setText("Ficheros en Google Drive:");
114         Thread t = new Thread(new Runnable() {
115
116             @Override
117             public void run() {
118                 try {
119                     final StringBuilder sb = new StringBuilder();
120
121                     listf = drive.files().list().execute();
122
123                     if (listf.getItems().isEmpty()) {
124                         sb.append("No hay ficheros");
125                     } else {
126                         sb.append("Ficheros de Drive:").append("\n");
127                     }
128
129                     runOnUiThread(new Runnable() {
130
131                         @Override
132                         public void run() {
133
134                             List<File> items = listf.getItems();
135
136                             item = new ArrayList<String>();
137
138                             for (File f : items) {
139                                 sb.append(f.getTitle()).append("\n");
140                                 item.add(f.getTitle());
141                             }
142
143                             Toast.makeText(
144                                     ExploradorDrive.this,
145                                     "Listado de ficheros de Google Drive cargado",
146                                     Toast.LENGTH_LONG).show();
147
148                             ArrayAdapter<String> fileList = new ArrayAdapter<String>(
149                                     ExploradorDrive.this, R.layout.fila, item);
150
151                             setListAdapter(fileList);
152                         }
153                     });
```

```
                } catch (UserRecoverableAuthIOException e) {
                    startActivityForResult(e.getIntent(), REQUEST_AUTHORIZATION);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
        t.start();
    }
    //On List Item Click saves selected
    protected void onListItemClick(ListView l, View v, int position, long id) {
        String selection = l.getItemAtPosition(position).toString();

        List<File> items = listf.getItems();
        item = new ArrayList<String>();

        for (File f : items) {

            if (f.getTitle() == selection) {
                fname = f.getTitle();
            }
        }
    }

    /** Download a specified file from the web */
    public void updateData() {

        final String file_url = fdownloadUrl;
        /* Define and configure the progress dialog */
        final ProgressDialog myProgress = new ProgressDialog(this);
        myProgress.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        myProgress.setMessage(fname);
        myProgress.setTitle(fname);
        /* Show the progress dialog. */
        myProgress.show();

        /* Download our file */

        new Thread(new Runnable() {

            public void run() {
                try {

                    HttpClient client = new DefaultHttpClient();
                    HttpGet get = new HttpGet(fdownloadUrl);
                    get.setHeader("Authorization", "Bearer " + token);
                    org.apache.http.HttpResponse response = client.execute(get);

                    // create the new connection
                    URL url = new URL(file_url);
                    HttpURLConnection urlConnection = (HttpURLConnection) url
                            .openConnection();

                    // set up some things on the connection
                    urlConnection.setRequestMethod("GET");
                    urlConnection.setDoOutput(true);

                    // and connect!
                    urlConnection.connect();

                    // set the path where we want to save the file
                    // in this case, going to save it on the root directory of
                    // the
                    // sd card.
                    java.io.File SDCardRoot = getExternalFilesDir(null);
                    // create a new file, specifying the path, and the filename
                    // which we want to save the file as.
                    java.io.File file = new java.io.File(SDCardRoot, fname);

                    // this will be used to write the downloaded data into the
                    // file we created
                    FileOutputStream fileOutput = new FileOutputStream(file);

                    // this will be used in reading the data from the internet
                    // InputStream inputStream = urlConnection.getInputStream();
                    InputStream inputStream = response.getEntity().getContent();

                    // this is the total size of the file
                    int totalSize = urlConnection.getContentLength();
                    myProgress.setMax(totalSize);
                    // variable to store total downloaded bytes
                    int downloadedSize = 0;
```

```
                int downloadedSize = 0;

                // create a buffer...
                byte[] buffer = new byte[1024];
                int bufferLength = 0; // used to store a temporary size of
                                      // the buffer
                int progress = 0;
                // now, read through the input buffer and write the contents
                // to the file
                while ((bufferLength = inputStream.read(buffer)) > 0) {
                    // add the data in the buffer to the file in the file
                    // output stream (the file on the sd card
                    fileOutput.write(buffer, 0, bufferLength);
                    // add up the size so we know how much is downloaded
                    downloadedSize += bufferLength;
                    // Here we update the progress
                    progress = downloadedSize;
                    myProgress.setProgress(progress);
                }
                // close the output stream when done
                fileOutput.close();
                myProgress.dismiss();
                // catch some possible errors...
            } catch (MalformedURLException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }).start();
}

public boolean onOptionsItemSelected(MenuItem item) {
    try {
        finish();

        return super.onOptionsItemSelected(item);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}

public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.play2cloud_back_menu, menu);
    return true;
}
}
```

7. ExploradorDropbox.java

```
package com.example.copy2cloud;

import android.content.DialogInterface;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;

import android.app.AlertDialog;
import android.app.ListActivity;
import android.text.method.ScrollingMovementMethod;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.dropbox.client2.DropboxAPI;
import com.dropbox.client2.DropboxAPI.Entry;
import com.dropbox.client2.android.AndroidAuthSession;
import com.dropbox.client2.exception.DropboxException;
import com.dropbox.client2.session.AccessTokenPair;
import com.dropbox.client2.session.AppKeyPair;
import com.dropbox.client2.session.Session.AccessType;
//It makes a Simple Dropbox File Explorer in APP Folder in Dropbox Service
public class ExploradorDropbox extends ListActivity {

    private DropboxAPI<AndroidAuthSession> mDBApi;
    final static String APP_KEY = "[redacted]";
    final static String APP_SECRET = "[redacted]";
    final static AccessType ACCESS_TYPE = AccessType.APP_FOLDER;

    String fichero = null;
    String path = "/";
    String path_back = "/";

    private List<String> pathBack = new ArrayList<String>();
    private List<String> item = null;
    private int pathBackCounter = 0;

    SharedPreferences prefs;

    private TextView output;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.explorador);

        output = (TextView) findViewById(R.id.path);
        output.setMovementMethod(new ScrollingMovementMethod());

        prefs = PreferenceManager.getDefaultSharedPreferences(this);

        String dropbox_key = prefs.getString("dropbox_key", "");
        String dropbox_secret = prefs.getString("dropbox_secret", "");

        if (dropbox_key.length() > 0 && dropbox_secret.length() > 0) {
            AccessTokenPair access = new AccessTokenPair(dropbox_key,
                    dropbox_secret);
            AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
```

```
72              AndroidAuthSession session = new AndroidAuthSession(appKeys,
73                      ACCESS_TYPE);
74              mDBApi = new DropboxAPI<AndroidAuthSession>(session);
75              mDBApi.getSession().setAccessTokenPair(access);
76          }
77          AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
78          AndroidAuthSession session = new AndroidAuthSession(appKeys,
79                  ACCESS_TYPE);
80
81          if (mDBApi == null) {
82              mDBApi = new DropboxAPI<AndroidAuthSession>(session);
83          }
84          mDBApi.getSession().startAuthentication(ExploradorDropbox.this);
85          listFilesDropbox(path);
86      }
87
88      /**
89       * Downloads a File List from Dropbox and
90       * put into a FileList Object.
91       */
92      public void listFilesDropbox(final String ruta) {
93          output.setText("Ficheros en Dropbox:");
94          Thread t = new Thread(new Runnable() {
95
96              @Override
97              public void run() {
98                  try {
99                      final StringBuilder sb = new StringBuilder();
100                     final Entry listf = mDBApi.metadata(ruta, 100, null, true,
101                             null);
102
103                     if (listf.contents.isEmpty()) {
104                         sb.append("No hay ficheros");
105                     } else {
106                         sb.append("Ficheros de Dropbox:").append("\n");
107                     }
108
109                     runOnUiThread(new Runnable() {
110                         @Override
111                         public void run() {
112                             item = new ArrayList<String>();
113                             item.add("..");
114                             for (Entry e : listf.contents) {
115                                 if (!e.isDeleted) {
116                                     if (e.isDir == true) {
117                                         item.add(String.valueOf(e.fileName())
118                                                 + "/");
119                                     } else {
120                                         item.add(String.valueOf(e.fileName()));
121                                     }
122                                 }
123                             }
124                             // Toast.makeText(ExploradorDropbox.this,
125                             // "Listado de ficheros de Dropbox cargado",
126                             // Toast.LENGTH_LONG).show();
127                             ArrayAdapter<String> fileList = new ArrayAdapter<String>(
128                                     ExploradorDropbox.this, R.layout.fila, item);
129
130                             setListAdapter(fileList);
131                         }
132                     });
133                 } catch (DropboxException e1) {
134                     // TODO Auto-generated catch block
135                     e1.printStackTrace();
136                 }
137             }
138         });
139         t.start();
140     }
141     //Navigation options on Item List Click, If Its a directory It opens
142     //If it's a File, It downloads
143     protected void onListItemClick(ListView l, View v, int position, long id) {
144         final String selection = l.getItemAtPosition(position).toString();
145
146         if (selection.contains("/") && selection != ("")) {
147             pathBack.add(path); // añadimos ruta vieja a pathBack
148             path_back = pathBack.get(pathBackCounter); // añadimos ruta vieja a
149                                                        // path_back almacenada
150                                                        // en el Array pathBack
151             pathBackCounter = pathBackCounter + 1; // aumentamos puntero del
152                                                    // array
```

```
153
154             path = path + selection;
155             listFilesDropbox(path);
156
157         } else if (selection.contains("..") && path != "/"
158                 && pathBackCounter > 0) {
159             pathBackCounter = pathBackCounter - 1;
160             path = pathBack.get(pathBackCounter);
161             pathBack.remove(pathBackCounter);
162
163             listFilesDropbox(path);
164
165         } else if (selection != "..") {
166
167             new AlertDialog.Builder(this)
168                     .setTitle("Atención: Descarga de Fichero")
169                     .setMessage(
170                             "Estás seguro de descargar el fichero " + selection)
171                     .setPositiveButton("Descargar",
172                             new DialogInterface.OnClickListener() {
173                                 public void onClick(DialogInterface dialog,
174                                         int which) {
175                                     download(path, selection);
176                                 }
177                             })
178                     .setNegativeButton("Cancelar",
179                             new DialogInterface.OnClickListener() {
180                                 public void onClick(DialogInterface dialog,
181                                         int which) {
182                                     // do nothing
183                                 }
184                             }).show();
185         }
186     }
187
188     public void download(String path, String file) {
189         try {
190             File output = new File(path + file);
191             OutputStream out = new FileOutputStream(output);
192             mDBApi.getFile(path + file, null, out, null);
193
194             Toast.makeText(
195                     ExploradorDropbox.this,
196                     "Fichero: " + file + " descargado exitosamente en: " + path,
197                     Toast.LENGTH_LONG).show();
198             finish();
199         } catch (FileNotFoundException e) {
200             e.printStackTrace();
201         } catch (DropboxException e) {
202             e.printStackTrace();
203         }
204
205     }
206
207     @Override
208     protected void onResume() {
209         super.onResume();
210         if (mDBApi != null && mDBApi.getSession().authenticationSuccessful()) {
211             try {
212                 mDBApi.getSession().finishAuthentication();
213                 AccessTokenPair tokens = mDBApi.getSession()
214                         .getAccessTokenPair();
215                 Editor editor = prefs.edit();
216                 editor.putString("dropbox_key", tokens.key);
217                 editor.putString("dropbox_secret", tokens.secret);
218                 editor.commit();
219             } catch (IllegalStateException e) {
220                 Log.i("DbAuthLog", "Error authenticating", e);
221             }
222         }
223     }
224
225     public boolean onOptionsItemSelected(MenuItem item) {
226         try {
227             finish();
228
229             return super.onOptionsItemSelected(item);
230         } catch (Exception e) {
231             e.printStackTrace();
232         }
233         return false;
234     }
235
236     public boolean onCreateOptionsMenu(Menu menu) {
237         // Inflate the menu; this adds items to the action bar if it is present.
238         getMenuInflater().inflate(R.menu.play2cloud_back_menu, menu);
239         return true;
240     }
241 }
```

8. ExploradorDropboxP2C.java

```
package com.example.copy2cloud;

import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.List;

import android.annotation.SuppressLint;
import android.app.AlertDialog;
import android.app.ListActivity;
import android.text.method.ScrollingMovementMethod;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.dropbox.client2.DropboxAPI;
import com.dropbox.client2.DropboxAPI.DropboxLink;
import com.dropbox.client2.DropboxAPI.Entry;
import com.dropbox.client2.android.AndroidAuthSession;
import com.dropbox.client2.exception.DropboxException;
import com.dropbox.client2.session.AccessTokenPair;
import com.dropbox.client2.session.AppKeyPair;
import com.dropbox.client2.session.Session.AccessType;

public class ExploradorDropboxP2C extends ListActivity {

    private DropboxAPI<AndroidAuthSession> mDBApi;
    final static String APP_KEY = "[redacted]";
    final static String APP_SECRET = "[redacted]";
    final static AccessType ACCESS_TYPE = AccessType.APP_FOLDER;

    String fichero = null;
    @SuppressLint("SdCardPath")
    String path = "/mnt/sdcard/P2C/";
    String path_back = "/";

    private List<String> item = null;
    private List<String> shareList = new ArrayList<String>();

    SharedPreferences prefs;

    private TextView output;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.explorador);

        output = (TextView) findViewById(R.id.path);
        output.setMovementMethod(new ScrollingMovementMethod());

        prefs = PreferenceManager.getDefaultSharedPreferences(this);

        String dropbox_key = prefs.getString("dropbox_key", "");
        String dropbox_secret = prefs.getString("dropbox_secret", "");
        if (dropbox_key.length() > 0 && dropbox_secret.length() > 0) {
            AccessTokenPair access = new AccessTokenPair(dropbox_key,
                    dropbox_secret);
            AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
            AndroidAuthSession session = new AndroidAuthSession(appKeys,
                    ACCESS_TYPE);
            mDBApi = new DropboxAPI<AndroidAuthSession>(session);
```

```
81              mDBApi.getSession().setAccessTokenPair(access);
82          }
83          AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
84          AndroidAuthSession session = new AndroidAuthSession(appKeys,
85                  ACCESS_TYPE);
86
87          if (mDBApi == null) {
88              mDBApi = new DropboxAPI<AndroidAuthSession>(session);
89          }
90          mDBApi.getSession().startAuthentication(ExploradorDropboxP2C.this);
91          createFolder();
92          listFilesDropbox(path);
93      }
94
95      /**
96       * Descarga una lista de ficheros desde Dropbox y los presenta en un
97       * FileList.
98       */
99      public void listFilesDropbox(final String ruta) {
100         output.setText("Ficheros en Dropbox:");
101         Thread t = new Thread(new Runnable() {
102
103             @Override
104             public void run() {
105                 try {
106                     final StringBuilder sb = new StringBuilder();
107                     final Entry listf = mDBApi.metadata(ruta, 100, null, true,
108                             null);
109
110                     if (listf.contents.isEmpty()) {
111                         sb.append("No hay ficheros");
112                     } else {
113                         sb.append("Ficheros de Dropbox:").append("\n");
114
115                     }
116
117                     runOnUiThread(new Runnable() {
118                         @Override
119                         public void run() {
120                             item = new ArrayList<String>();
121                             item.add("..");
122                             for (Entry e : listf.contents) {
123                                 if (!e.isDeleted) {
124                                     if (e.isDir == true) {
125                                         item.add(String.valueOf(e.fileName())
126                                                 + "/");
127                                     } else {
128                                         item.add(String.valueOf(e.fileName()));
129                                         try {
130                                             DropboxLink DBLink = mDBApi.share(path
131                                                     + String.valueOf(e
132                                                             .fileName()));
133                                             String shareAddress = getShareURL(
134                                                     DBLink.url)
135                                                     .replaceFirst(
136                                                             "https://www",
137                                                             "https://dl");
138
139                                             shareList.add(shareAddress);
140
141                                         } catch (DropboxException e1) {
142                                             e1.printStackTrace();
143                                         }
144                                     }
145                                 }
146                             }
147                             ArrayAdapter<String> fileList = new ArrayAdapter<String>(
148                                     ExploradorDropboxP2C.this, R.layout.fila,
149                                     item);
150
151                             setListAdapter(fileList);
152                         }
153                     });
154                 } catch (DropboxException e1) {
155                     e1.printStackTrace();
156                 }
157             }
```

```
        });
        t.start();
    }

    // Método para obtener URL pública de Dropbox
    String getShareURL(String strURL) {
        URLConnection conn = null;

        try {
            URL inputURL = new URL(strURL);
            conn = inputURL.openConnection();

        } catch (MalformedURLException e) {

        } catch (IOException ioe) {

        }
        return conn.getHeaderField("location");
    }

    protected void onListItemClick(ListView l, View v, final int position,
            long id) {
        final String selection = l.getItemAtPosition(position).toString();
        final Intent i = new Intent(getApplicationContext(), Play2Cloud.class);

        if (selection != "..") {

            new AlertDialog.Builder(this)
                    .setTitle("Atención: Reprucción de Stream de Fichero")
                    .setMessage(
                            "Estás seguro de reproducir el fichero "
                                    + selection)
                    .setPositiveButton("Reproducir",
                            new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog,
                                        int which) {

                                    Toast.makeText(
                                            ExploradorDropboxP2C.this,
                                            "Cacheando fichero... Espere unos instantes...",
                                            Toast.LENGTH_LONG).show();
                                    i.putExtra("direccion",
                                            shareList.get(position - 1)
                                                    .toString());
                                    i.putExtra("nombre",
                                            item.get(position)
                                                    .toString());
                                    startActivity(i);
                                }
                            })
                    .setNegativeButton("Cancelar",
                            new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog,
                                        int which) {
                                }
                            }).show();
        }
    }

    public void createFolder() {
        try {
            mDBApi.createFolder(path);

        } catch (DropboxException e) {
            e.printStackTrace();

        }
    }

    public void download(String path, String file) {
        try {
            File output = new File(path + file);
            OutputStream out = new FileOutputStream(output);
            mDBApi.getFile(path + file, null, out, null);

            Toast.makeText(
                    ExploradorDropboxP2C.this,
```

```
                "Fichero: " + file + " descargado exitosamente en: " + path,
                Toast.LENGTH_LONG).show();
            finish();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (DropboxException e) {
            e.printStackTrace();
        }
    }

    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.play2cloud_menu, menu);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.menu_subir_mp3:

            Intent i = new Intent(getApplicationContext(),
                    Files2CloudUpload.class);
            Toast.makeText(
                    ExploradorDropboxP2C.this,
                    "Crea una carpeta en /sdcard/P2C/ y pon en esta los ficheros mp3 a subir",
                    Toast.LENGTH_LONG).show();
            startActivity(i);
            finish();

            return true;

        default:
            return super.onOptionsItemSelected(item);
        }
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (mDBApi != null && mDBApi.getSession().authenticationSuccessful()) {
            try {
                mDBApi.getSession().finishAuthentication();
                AccessTokenPair tokens = mDBApi.getSession()
                        .getAccessTokenPair();
                Editor editor = prefs.edit();
                editor.putString("dropbox_key", tokens.key);
                editor.putString("dropbox_secret", tokens.secret);
                editor.commit();
            } catch (IllegalStateException e) {
                Log.i("DbAuthLog", "Error authenticating", e);
            }
        }
    }
}
```

9. ExploradorDropboxW2C.java

```
package com.example.copy2cloud;

import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import java.util.List;

import android.annotation.SuppressLint;
import android.app.AlertDialog;
import android.app.ListActivity;
import android.text.method.ScrollingMovementMethod;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.dropbox.client2.DropboxAPI;
import com.dropbox.client2.DropboxAPI.DropboxLink;
import com.dropbox.client2.DropboxAPI.Entry;
import com.dropbox.client2.android.AndroidAuthSession;
import com.dropbox.client2.exception.DropboxException;
import com.dropbox.client2.session.AccessTokenPair;
import com.dropbox.client2.session.AppKeyPair;
import com.dropbox.client2.session.Session.AccessType;

public class ExploradorDropboxW2C extends ListActivity {

    private DropboxAPI<AndroidAuthSession> mDBApi;
    final static String APP_KEY = "[redacted]";
    final static String APP_SECRET = "[redacted]";
    final static AccessType ACCESS_TYPE = AccessType.APP_FOLDER;

    String fichero = null;
    @SuppressLint("SdCardPath")
    String path = "/mnt/sdcard/W2C/";
    String path_back = "/";

    private List<String> item = null;
    private List<String> shareList = new ArrayList<String>();

    SharedPreferences prefs;

    private TextView output;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.explorador);

        output = (TextView) findViewById(R.id.path);
        output.setMovementMethod(new ScrollingMovementMethod());

        prefs = PreferenceManager.getDefaultSharedPreferences(this);

        String dropbox_key = prefs.getString("dropbox_key", "");
        String dropbox_secret = prefs.getString("dropbox_secret", "");
        if (dropbox_key.length() > 0 && dropbox_secret.length() > 0) {
            AccessTokenPair access = new AccessTokenPair(dropbox_key,
                    dropbox_secret);
```

```
77              AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
78              AndroidAuthSession session = new AndroidAuthSession(appKeys,
79                      ACCESS_TYPE);
80              mDBApi = new DropboxAPI<AndroidAuthSession>(session);
81              mDBApi.getSession().setAccessTokenPair(access);
82          }
83          AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
84          AndroidAuthSession session = new AndroidAuthSession(appKeys,
85                  ACCESS_TYPE);
86
87          if (mDBApi == null) {
88              mDBApi = new DropboxAPI<AndroidAuthSession>(session);
89          }
90          mDBApi.getSession().startAuthentication(ExploradorDropboxW2C.this);
91          createFolder();
92          listFilesDropbox(path);
93      }
94
95      /**
96       * Descarga una lista de ficheros desde Dropbox y los presenta en un
97       * FileList.
98       */
99      public void listFilesDropbox(final String ruta) {
100         output.setText("Ficheros en Dropbox:");
101         Thread t = new Thread(new Runnable() {
102
103             @Override
104             public void run() {
105                 try {
106                     final StringBuilder sb = new StringBuilder();
107                     final Entry listf = mDBApi.metadata(ruta, 100, null, true,
108                             null);
109
110                     if (listf.contents.isEmpty()) {
111                         sb.append("No hay ficheros");
112                     } else {
113                         sb.append("Ficheros de Dropbox:").append("\n");
114
115                     }
116
117                     runOnUiThread(new Runnable() {
118                         @Override
119                         public void run() {
120                             item = new ArrayList<String>();
121                             item.add("..");
122                             for (Entry e : listf.contents) {
123                                 if (!e.isDeleted) {
124                                     if (e.isDir == true) {
125                                         item.add(String.valueOf(e.fileName())
126                                                 + "/");
127                                     } else {
128                                         item.add(String.valueOf(e.fileName()));
129
130                                         try {
131                                             DropboxLink DBLink = mDBApi.share(path
132                                                     + String.valueOf(e
133                                                             .fileName()));
134                                             String shareAddress = getShareURL(
135                                                     DBLink.url)
136                                                     .replaceFirst(
137                                                             "https://www",
138                                                             "https://dl");
139
140                                             shareList.add(shareAddress);
141
142                                         } catch (DropboxException e1) {
143                                             e1.printStackTrace();
144                                         }
145                                     }
146                                 }
147                             }
148                             ArrayAdapter<String> fileList = new ArrayAdapter<String>(
149                                     ExploradorDropboxW2C.this, R.layout.fila,
150                                     item);
151
152                             setListAdapter(fileList);
```

```
                    }
                });
            } catch (DropboxException e1) {
                e1.printStackTrace();
            }
        }
    });
    t.start();
}

// Método para obtener URL pública de Dropbox
String getShareURL(String strURL) {
    URLConnection conn = null;

    try {
        URL inputURL = new URL(strURL);
        conn = inputURL.openConnection();

    } catch (MalformedURLException e) {

    } catch (IOException ioe) {

    }
    return conn.getHeaderField("location");
}

protected void onListItemClick(ListView l, View v, final int position,
        long id) {
    final String selection = l.getItemAtPosition(position).toString();
    final Intent i = new Intent(getApplicationContext(), Write2Cloud.class);

    if (selection != "..") {

        new AlertDialog.Builder(this)
                .setTitle("Atención: Abrir Nota")
                .setMessage("Desea abrir el fichero de notas " + selection)
                .setPositiveButton("Abrir",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                    int which) {

                                Toast.makeText(
                                        ExploradorDropboxW2C.this,
                                        "Descargando fichero... Espere unos instantes...",
                                        Toast.LENGTH_LONG).show();
                                i.putExtra("direccion",
                                        shareList.get(position - 1)
                                                .toString());
                                i.putExtra("nombre", item.get(position)
                                        .toString());
                                startActivity(i);
                            }
                        })
                .setNegativeButton("Cancelar",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                    int which) {
                            }
                        }).show();
    }
}

public void createFolder() {
    try {
        mDBApi.createFolder(path);

    } catch (DropboxException e) {
        e.printStackTrace();
    }
}

public void download(String path, String file) {
    try {
        File output = new File(path + file);
        OutputStream out = new FileOutputStream(output);
        mDBApi.getFile(path + file, null, out, null);

        Toast.makeText(
                ExploradorDropboxW2C.this,
                "Fichero: " + file + " descargado exitosamente en: " + path,
                Toast.LENGTH_LONG).show();
        finish();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
```

```
237         } catch (DropboxException e) {
238             e.printStackTrace();
239         }
240 
241     }
242 
243     public boolean onCreateOptionsMenu(Menu menu) {
244         // Inflate the menu; this adds items to the action bar if it is present.
245         getMenuInflater().inflate(R.menu.dropbox_menu, menu);
246         return true;
247     }
248 
249     public boolean onOptionsItemSelected(MenuItem item) {
250         switch (item.getItemId()) {
251         case R.id.menu_open:
252 
253             Intent i = new Intent(getApplicationContext(), Write2Cloud.class);
254             i.putExtra("direccion", "");
255             i.putExtra("nombre", "nueva_nota.txt");
256             startActivity(i);
257             finish();
258 
259             return true;
260 
261         default:
262             return super.onOptionsItemSelected(item);
263         }
264 
265     }
266 
267     @Override
268     protected void onResume() {
269         super.onResume();
270         if (mDBApi != null && mDBApi.getSession().authenticationSuccessful()) {
271             try {
272                 mDBApi.getSession().finishAuthentication();
273                 AccessTokenPair tokens = mDBApi.getSession()
274                         .getAccessTokenPair();
275                 Editor editor = prefs.edit();
276                 editor.putString("dropbox_key", tokens.key);
277                 editor.putString("dropbox_secret", tokens.secret);
278                 editor.commit();
279             } catch (IllegalStateException e) {
280                 Log.i("DbAuthLog", "Error authenticating", e);
281             }
282         }
283     }
284 }
```

10. Files2Cloud.java

```java
package com.example.copy2cloud;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import com.example.copy2cloud.R;

public class Files2Cloud extends Activity {
    /** Called when the activity is first created. */

    int request_code = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.files2cloud_layout);

        Button btn_files2cloud_upload = (Button) findViewById(R.id.files2cloud2);
        Button btn_files2cloud_download = (Button) findViewById(R.id.btn_conctats2cloud);

        // Listening Files2Cloud button click
        btn_files2cloud_upload.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                // Launching Files2CloudUpload Class
                Intent i = new Intent(getApplicationContext(),
                        Files2CloudUpload.class);
                startActivity(i);
            }
        });

        // Listening Files2Cloud button click
        btn_files2cloud_download.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                // Launching Files2CloudDownload Class
                Intent i = new Intent(getApplicationContext(),
                        Files2CloudDownload.class);
                startActivity(i);

            }

        });
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        try {
            finish();

            return super.onOptionsItemSelected(item);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.play2cloud_back_menu, menu);
        return true;
    }
}
```

11. Files2CloudDownload.java

```java
package com.example.copy2cloud;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import com.example.copy2cloud.R;

public class Files2CloudDownload extends Activity {
    /** Called when the activity is first created. */

    int request_code = 1;
    private TextView boxTexto;
    private String fichero;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.files2cloud_download_layout);

        Button btn_files2cloud = (Button) findViewById(R.id.files2cloud_gd);
        Button btn_files2cloud2 = (Button) findViewById(R.id.files2cloud_dp);

        boxTexto = (TextView) findViewById(R.id.tvTexto);

        // Listening Files2Cloud button click
        btn_files2cloud.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                // Launching ExploradorDrive Class
                Intent i = new Intent(getApplicationContext(),
                        ExploradorDrive.class);
                startActivityForResult(i, request_code);
            }
        });

        // Listening Files2Cloud2 button click
        btn_files2cloud2.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                // Launching ExploradorDropbox Class
                Intent i = new Intent(getApplicationContext(),
                        ExploradorDropbox.class);
                startActivityForResult(i, request_code);
            }

        });
    }

    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if ((requestCode == request_code) && (resultCode == RESULT_OK)) {
            fichero = data.getDataString();
            boxTexto.setText(fichero);
        }
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        try {
            finish();

            return super.onOptionsItemSelected(item);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.files2cloud_back_menu, menu);
        return true;
    }
}
```

12. Files2CloudUpload.java

```
package com.example.copy2cloud;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.example.copy2cloud.R;

public class Files2CloudUpload extends Activity {
    /** Called when the activity is first created. */

    int request_code = 1;
    private TextView boxTexto;
    private String fichero;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.files2cloud_upload_layout);

        Button btn_files2cloud = (Button) findViewById(R.id.btn_conctats2cloud);
        Button btn_files2cloud2 = (Button) findViewById(R.id.files2cloud_gd);
        Button btn_files2cloud3 = (Button) findViewById(R.id.files2cloud_dp);

        boxTexto = (TextView) findViewById(R.id.tvTexto);

        // Listening Files2Cloud button click
        btn_files2cloud.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                // Launching Camera2Cloud Screen
                Intent i = new Intent(getApplicationContext(), Explorador.class);
                startActivityForResult(i, request_code);
            }
        });

        // Listening Files2Cloud button click
        btn_files2cloud2.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                // Launching Camera2Cloud Screen
                Intent i = new Intent(getApplicationContext(), FileUpload.class);

                if (fichero != null) {
                    i.putExtra("direccion", fichero);
                    startActivity(i);
                    Toast.makeText(
                            Files2CloudUpload.this,
                            "Subiendo fichero: " + fichero
                                    + ", espere unos instantes...",
                            Toast.LENGTH_LONG).show();
                } else {
                    Toast.makeText(Files2CloudUpload.this,
                            "Selecciona primero un fichero que subir!!",
                            Toast.LENGTH_LONG).show();
                }
            }

        });

        btn_files2cloud3.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View view) {
                Intent i = new Intent(getApplicationContext(),
                        FileUploadDropbox.class);

```

```
76                  // Launching Camera2Cloud Screen
77                  if (fichero != null) {
78                      i.putExtra("direccion", fichero);
79                      startActivity(i);
80                      Toast.makeText(
81                              Files2CloudUpload.this,
82                              "Subiendo fichero: " + fichero
83                                      + ", espere unos instantes...",
84                              Toast.LENGTH_LONG).show();
85
86                  } else {
87                      Toast.makeText(Files2CloudUpload.this,
88                              "Selecciona primero un fichero que subir!!",
89                              Toast.LENGTH_LONG).show();
90
91                  }
92
93              }
94
95          });
96      }
97
98      protected void onActivityResult(int requestCode, int resultCode, Intent data) {
99          if ((requestCode == request_code) && (resultCode == RESULT_OK)) {
100             fichero = data.getDataString();
101             boxTexto.setText(fichero);
102         }
103     }
104
105     public boolean onOptionsItemSelected(MenuItem item) {
106         try {
107             finish();
108
109             return super.onOptionsItemSelected(item);
110         } catch (Exception e) {
111             e.printStackTrace();
112         }
113         return false;
114     }
115
116     public boolean onCreateOptionsMenu(Menu menu) {
117         // Inflate the menu; this adds items to the action bar if it is present.
118         getMenuInflater().inflate(R.menu.files2cloud_back_menu, menu);
119         return true;
120     }
121 }
```

13. FileUpload.java

```java
package com.example.copy2cloud;

import java.io.IOException;

import android.accounts.AccountManager;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

import com.google.api.client.extensions.android.http.AndroidHttp;
import com.google.api.client.googleapis.extensions.android.gms.auth.GoogleAccountCredential;
import com.google.api.client.googleapis.extensions.android.gms.auth.UserRecoverableAuthIOException;
import com.google.api.client.http.FileContent;
import com.google.api.client.json.gson.GsonFactory;
import com.google.api.services.drive.Drive;
import com.google.api.services.drive.DriveScopes;
import com.google.api.services.drive.model.File;

public class FileUpload extends Activity {
    static final int REQUEST_ACCOUNT_PICKER = 1;
    static final int REQUEST_AUTHORIZATION = 2;
    static final int UPLOAD_FILE = 3;

    private static Drive service;
    private String fichero;
    private GoogleAccountCredential credential;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Bundle bundle = getIntent().getExtras();
        fichero = bundle.getString("direccion");

        credential = GoogleAccountCredential.usingOAuth2(this,
                DriveScopes.DRIVE);
        startActivityForResult(credential.newChooseAccountIntent(),
                REQUEST_ACCOUNT_PICKER);
    }

    @Override
    protected void onActivityResult(final int requestCode,
            final int resultCode, final Intent data) {
        switch (requestCode) {
        case REQUEST_ACCOUNT_PICKER:
            if (resultCode == RESULT_OK && data != null
                    && data.getExtras() != null) {
                String accountName = data
                        .getStringExtra(AccountManager.KEY_ACCOUNT_NAME);

                if (accountName != null) {
                    credential.setSelectedAccountName(accountName);
                    service = getDriveService(credential);

                    saveFileToDrive();
                }
            }
            break;
        }
    }

    private void saveFileToDrive() {
        Thread thread = new Thread(new Runnable() {

            @Override
            public void run() {

                Uri.fromFile(new java.io.File(fichero));
                java.io.File fileContent = new java.io.File(fichero);
                FileContent mediaContent = new FileContent("text/plain",
                        fileContent);

                File body = new File();
```

```
78                  body.setTitle(fileContent.getName());
79                  body.setMimeType("text/plain");
80
81                  try {
82                      File file = service.files().insert(body, mediaContent)
83                              .execute();
84
85                      if (file != null) { // file
86                          showToast("Fichero subido: " + file.getTitle());
87                          finish();
88                      }
89                  } catch (UserRecoverableAuthIOException e) {
90                      startActivityForResult(e.getIntent(), REQUEST_AUTHORIZATION);
91                  } catch (IOException e) {
92                      e.printStackTrace();
93                  }
94              }
95          });
96          thread.start();
97      }
98
99      private Drive getDriveService(GoogleAccountCredential credential) {
100         return new Drive.Builder(AndroidHttp.newCompatibleTransport(),
101                 new GsonFactory(), credential).build();
102     }
103
104     public void showToast(final String toast) {
105         runOnUiThread(new Runnable() {
106             @Override
107             public void run() {
108                 Toast.makeText(getApplicationContext(), toast,
109                         Toast.LENGTH_LONG).show();
110             }
111         });
112     }
113
114     public boolean onOptionsItemSelected(MenuItem item) {
115         try {
116             finish();
117
118             return super.onOptionsItemSelected(item);
119         } catch (Exception e) {
120             e.printStackTrace();
121         }
122         return false;
123     }
124
125     public boolean onCreateOptionsMenu(Menu menu) {
126         // Inflate the menu; this adds items to the action bar if it is present.
127         getMenuInflater().inflate(R.menu.play2cloud_back_menu, menu);
128         return true;
129     }
130 }
```

14. FileUploadDropbox.java

```java
package com.example.copy2cloud;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import com.dropbox.client2.DropboxAPI;
import com.dropbox.client2.android.AndroidAuthSession;
import com.dropbox.client2.exception.DropboxException;
import com.dropbox.client2.session.AccessTokenPair;
import com.dropbox.client2.session.AppKeyPair;
import com.dropbox.client2.session.Session.AccessType;

public class FileUploadDropbox extends Activity {
    private DropboxAPI<AndroidAuthSession> mDBApi;
    final static String APP_KEY = "[illegible]";
    final static String APP_SECRET = "[illegible]";
    final static AccessType ACCESS_TYPE = AccessType.APP_FOLDER;
    String fichero = null;

    SharedPreferences prefs;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // setContentView(R.layout.main);
        prefs = PreferenceManager.getDefaultSharedPreferences(this);

        Bundle bundle = getIntent().getExtras();
        fichero = bundle.getString("direccion");

        String dropbox_key = prefs.getString("dropbox_key", "");
        String dropbox_secret = prefs.getString("dropbox_secret", "");
        if (dropbox_key.length() > 0 && dropbox_secret.length() > 0) {
            AccessTokenPair access = new AccessTokenPair(dropbox_key,
                    dropbox_secret);
            AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
            AndroidAuthSession session = new AndroidAuthSession(appKeys,
                    ACCESS_TYPE);
            mDBApi = new DropboxAPI<AndroidAuthSession>(session);
            mDBApi.getSession().setAccessTokenPair(access);
        }

        AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
        AndroidAuthSession session = new AndroidAuthSession(appKeys,
                ACCESS_TYPE);

        if (mDBApi == null) {
            mDBApi = new DropboxAPI<AndroidAuthSession>(session);
        }
        mDBApi.getSession().startAuthentication(FileUploadDropbox.this);
        upload();
    }

    public void upload() {

        try {
            File file = new File(fichero);
            FileInputStream in = new FileInputStream(file);
            mDBApi.putFileOverwrite(fichero, in, file.length(), null);

            Toast.makeText(FileUploadDropbox.this,
                    "Fichero: " + fichero + " subido exitosamente!",
                    Toast.LENGTH_LONG).show();
            finish();
```

```
79          } catch (IOException e) {
80              e.printStackTrace();
81          } catch (DropboxException e) {
82              e.printStackTrace();
83          }
84
85      }
86
87      @SuppressLint("SdCardPath")
88      public void download() {
89          try {
90              File output = new File("/mnt/sdcard/test.txt");
91              OutputStream out = new FileOutputStream(output);
92              mDBApi.getFile("/test.txt", null, out, null);
93          } catch (FileNotFoundException e) {
94              e.printStackTrace();
95          } catch (DropboxException e) {
96              e.printStackTrace();
97          }
98
99      }
100
101     @Override
102     protected void onResume() {
103         super.onResume();
104         if (mDBApi != null && mDBApi.getSession().authenticationSuccessful()) {
105             try {
106                 mDBApi.getSession().finishAuthentication();
107                 AccessTokenPair tokens = mDBApi.getSession()
108                         .getAccessTokenPair();
109                 Editor editor = prefs.edit();
110                 editor.putString("dropbox_key", tokens.key);
111                 editor.putString("dropbox_secret", tokens.secret);
112                 editor.commit();
113             } catch (IllegalStateException e) {
114                 Log.i("DbAuthLog", "Error authenticating", e);
115             }
116         }
117     }
118
119     public boolean onOptionsItemSelected(MenuItem item) {
120         try {
121             finish();
122
123             return super.onOptionsItemSelected(item);
124         } catch (Exception e) {
125             e.printStackTrace();
126         }
127         return false;
128     }
129
130     public boolean onCreateOptionsMenu(Menu menu) {
131         // Inflate the menu; this adds items to the action bar if it is present.
132         getMenuInflater().inflate(R.menu.play2cloud_back_menu, menu);
133         return true;
134     }
135 }
```

15. MenuLayout.java

```java
package com.example.copy2cloud;

import android.content.Context;
import android.util.AttributeSet;
import android.view.View;
import android.view.ViewGroup;

/**
 * Custom layout that arranges children in a grid-like manner, optimizing for
 * even horizontal and vertical whitespace.
 */
public class MenuLayout extends ViewGroup {

    private static final int UNEVEN_GRID_PENALTY_MULTIPLIER = 10;

    private int mMaxChildWidth = 0;
    private int mMaxChildHeight = 0;

    public MenuLayout(Context context) {
        super(context, null);
    }

    public MenuLayout(Context context, AttributeSet attrs) {
        super(context, attrs, 0);
    }

    public MenuLayout(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        mMaxChildWidth = 0;
        mMaxChildHeight = 0;

        // Measure once to find the maximum child size.

        int childWidthMeasureSpec = MeasureSpec.makeMeasureSpec(
                MeasureSpec.getSize(widthMeasureSpec), MeasureSpec.AT_MOST);
        int childHeightMeasureSpec = MeasureSpec.makeMeasureSpec(
                MeasureSpec.getSize(widthMeasureSpec), MeasureSpec.AT_MOST);

        final int count = getChildCount();
        for (int i = 0; i < count; i++) {
            final View child = getChildAt(i);
            if (child.getVisibility() == GONE) {
                continue;
            }

            child.measure(childWidthMeasureSpec, childHeightMeasureSpec);

            mMaxChildWidth = Math.max(mMaxChildWidth, child.getMeasuredWidth());
            mMaxChildHeight = Math.max(mMaxChildHeight,
                    child.getMeasuredHeight());
        }

        // Measure again for each child to be exactly the same size.

        childWidthMeasureSpec = MeasureSpec.makeMeasureSpec(mMaxChildWidth,
                MeasureSpec.EXACTLY);
        childHeightMeasureSpec = MeasureSpec.makeMeasureSpec(mMaxChildHeight,
                MeasureSpec.EXACTLY);

        for (int i = 0; i < count; i++) {
            final View child = getChildAt(i);
            if (child.getVisibility() == GONE) {
                continue;
            }

            child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
        }

        setMeasuredDimension(resolveSize(mMaxChildWidth, widthMeasureSpec),
                resolveSize(mMaxChildHeight, heightMeasureSpec));
    }

```

```
77      @Override
78      protected void onLayout(boolean changed, int l, int t, int r, int b) {
79          int width = r - l;
80          int height = b - t;
81
82          final int count = getChildCount();
83
84          // Calculate the number of visible children.
85          int visibleCount = 0;
86          for (int i = 0; i < count; i++) {
87              final View child = getChildAt(i);
88              if (child.getVisibility() == GONE) {
89                  continue;
90              }
91              ++visibleCount;
92          }
93
94          if (visibleCount == 0) {
95              return;
96          }
97
98          // Calculate what number of rows and columns will optimize for even
99          // horizontal and
100         // vertical whitespace between items. Start with a 1 x N grid, then try
101         // 2 x N, and so on.
102         int bestSpaceDifference = Integer.MAX_VALUE;
103         int spaceDifference;
104
105         // Horizontal and vertical space between items
106         int hSpace = 0;
107         int vSpace = 0;
108
109         int cols = 1;
110         int rows;
111
112         while (true) {
113             rows = (visibleCount - 1) / cols + 1;
114
115             hSpace = ((width - mMaxChildWidth * cols) / (cols + 1));
116             vSpace = ((height - mMaxChildHeight * rows) / (rows + 1));
117
118             spaceDifference = Math.abs(vSpace - hSpace);
119             if (rows * cols != visibleCount) {
120                 spaceDifference *= UNEVEN_GRID_PENALTY_MULTIPLIER;
121             }
122
123             if (spaceDifference < bestSpaceDifference) {
124                 // Found a better whitespace squareness/ratio
125                 bestSpaceDifference = spaceDifference;
126
127                 // If we found a better whitespace squareness and there's only 1
128                 // row, this is
129                 // the best we can do.
130                 if (rows == 1) {
131                     break;
132                 }
133             } else {
134                 // This is a worse whitespace ratio, use the previous value of
135                 // cols and exit.
136                 --cols;
137                 rows = (visibleCount - 1) / cols + 1;
138                 hSpace = ((width - mMaxChildWidth * cols) / (cols + 1));
139                 vSpace = ((height - mMaxChildHeight * rows) / (rows + 1));
140                 break;
141             }
142
143             ++cols;
144         }
145
146         // Lay out children based on calculated best-fit number of rows and
147         // cols.
148
149         // If we chose a layout that has negative horizontal or vertical space,
150         // force it to zero.
151         hSpace = Math.max(0, hSpace);
152         vSpace = Math.max(0, vSpace);
153
154         // Re-use width/height variables to be child width/height.
155         width = (width - hSpace * (cols + 1)) / cols;
156         height = (height - vSpace * (rows + 1)) / rows;
```

```
157
158         int left, top;
159         int col, row;
160         int visibleIndex = 0;
161         for (int i = 0; i < count; i++) {
162             final View child = getChildAt(i);
163             if (child.getVisibility() == GONE) {
164                 continue;
165             }
166
167             row = visibleIndex / cols;
168             col = visibleIndex % cols;
169
170             left = hSpace * (col + 1) + width * col;
171             top = vSpace * (row + 1) + height * row;
172
173             child.layout(left, top, (hSpace == 0 && col == cols - 1) ? r
174                     : (left + width), (vSpace == 0 && row == rows - 1) ? b
175                     : (top + height));
176             ++visibleIndex;
177         }
178     }
179 }
```

16. Play2Cloud.java

```
1  package com.example.copy2cloud;
2
3  import java.io.File;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.net.URL;
8  import java.net.URLConnection;
9  import java.util.ArrayList;
10 import java.util.HashMap;
11 import java.util.Random;
12
13 import android.annotation.SuppressLint;
14 import android.annotation.TargetApi;
15 import android.app.Activity;
16 import android.content.Intent;
17 import android.media.MediaPlayer;
18 import android.media.MediaPlayer.OnCompletionListener;
19 import android.os.Build;
20 import android.os.Bundle;
21 import android.os.Handler;
22 import android.os.StrictMode;
23 import android.util.Log;
24 import android.view.Menu;
25 import android.view.MenuItem;
26 import android.view.View;
27 import android.webkit.URLUtil;
28 import android.widget.ImageButton;
29 import android.widget.SeekBar;
30 import android.widget.TextView;
31 import android.widget.Toast;
32
33 @TargetApi(Build.VERSION_CODES.GINGERBREAD)
34 @SuppressLint("NewApi")
35 public class Play2Cloud extends Activity implements OnCompletionListener,
36         SeekBar.OnSeekBarChangeListener {
37
38     private static final String TAG = "VideoViewDemo";
39
40     private ImageButton btnPlay;
41     private ImageButton btnForward;
42     private ImageButton btnBackward;
43     private ImageButton btnNext;
44     private ImageButton btnPrevious;
45     private ImageButton btnPlaylist;
46     private ImageButton btnRepeat;
47     private ImageButton btnShuffle;
48     private SeekBar songProgressBar;
49     private TextView songTitleLabel;
50     private TextView songCurrentDurationLabel;
```

```
51      private TextView songTotalDurationLabel;
52      // Media Player
53      private MediaPlayer mp;
54      // Handler to update UI timer, progress bar etc,.
55      private Handler mHandler = new Handler();;
56      private Utilities utils;
57      private int seekForwardTime = 5000; // 5000 milliseconds
58      private int seekBackwardTime = 5000; // 5000 milliseconds
59      private int currentSongIndex = 0;
60      private boolean isShuffle = false;
61      private boolean isRepeat = false;
62      private ArrayList<HashMap<String, String>> songsList = new ArrayList<HashMap<String, String>>();
63      private String URL;
64      private String nombre;
65      private FileOutputStream out;
66
67      @TargetApi(Build.VERSION_CODES.GINGERBREAD)
68      @SuppressLint("NewApi")
69      @Override
70      public void onCreate(Bundle savedInstanceState) {
71
72          super.onCreate(savedInstanceState);
73          setContentView(R.layout.player);
74
75          StrictMode.ThreadPolicy ourPolicy = new StrictMode.ThreadPolicy.Builder()
76                  .permitAll().build();
77          StrictMode.setThreadPolicy(ourPolicy);
78
79          Bundle bundle = getIntent().getExtras();
80          URL = bundle.getString("direccion");
81          nombre = bundle.getString("nombre");
82
83          // All player buttons
84          btnPlay = (ImageButton) findViewById(R.id.btnPlay);
85          btnForward = (ImageButton) findViewById(R.id.btnForward);
86          btnBackward = (ImageButton) findViewById(R.id.btnBackward);
87          btnNext = (ImageButton) findViewById(R.id.btnNext);
88          btnPrevious = (ImageButton) findViewById(R.id.btnPrevious);
89          btnPlaylist = (ImageButton) findViewById(R.id.btnPlaylist);
90          btnRepeat = (ImageButton) findViewById(R.id.btnRepeat);
91          btnShuffle = (ImageButton) findViewById(R.id.btnShuffle);
92          songProgressBar = (SeekBar) findViewById(R.id.songProgressBar);
93          songTitleLabel = (TextView) findViewById(R.id.songTitle);
94          songCurrentDurationLabel = (TextView) findViewById(R.id.songCurrentDurationLabel);
95          songTotalDurationLabel = (TextView) findViewById(R.id.songTotalDurationLabel);
96
97          // Mediaplayer
98
99          try {
100             /**
101              * Play button click event plays a song and changes button to pause
102              * image pauses a song and changes button to play image
103              * */
104             mp = new MediaPlayer();
105             utils = new Utilities();
106
107             // Listeners
108             songProgressBar.setOnSeekBarChangeListener(this); // Important
109             mp.setOnCompletionListener(this); // Important
110
111             mp.start();
112
113             btnPlay.setOnClickListener(new View.OnClickListener() {
114
115                 @Override
116                 public void onClick(View arg0) {
117                     // check for already playing
118                     if (mp.isPlaying()) {
119                         if (mp != null) {
120                             mp.pause();
121                             // Changing button image to play button
122                             btnPlay.setImageResource(R.drawable.btn_play);
123                         }
124                     } else {
125                         // Resume song
126                         if (mp != null) {
127                             if (URL == null) {
128                                 Toast.makeText(
129                                         Play2Cloud.this,
130                                         "Selecciona primero un fichero mp3 a reproducir!!",
```

```
                                    Toast.LENGTH_LONG).show();
                        } else {
                            mp.start();
                        }
                        // Changing button image to pause button
                        btnPlay.setImageResource(R.drawable.btn_pause);
                    }
                }

            }
        });

        /**
         * Forward button click event Forwards song specified seconds
         * */
        btnForward.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View arg0) {
                // get current song position
                int currentPosition = mp.getCurrentPosition();
                // check if seekForward time is lesser than song duration
                if (currentPosition + seekForwardTime <= mp.getDuration()) {
                    // forward song
                    mp.seekTo(currentPosition + seekForwardTime);
                } else {
                    // forward to end position
                    mp.seekTo(mp.getDuration());
                }
            }
        });

        /**
         * Backward button click event Backward song to specified seconds
         * */
        btnBackward.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View arg0) {
                // get current song position
                int currentPosition = mp.getCurrentPosition();
                // check if seekBackward time is greater than 0 sec
                if (currentPosition - seekBackwardTime >= 0) {
                    // forward song
                    mp.seekTo(currentPosition - seekBackwardTime);
                } else {
                    // backward to starting position
                    mp.seekTo(0);
                }

            }
        });

        /**
         * Next button click event Plays next song by taking
         * currentSongIndex + 1
         * */
        btnNext.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View arg0) {
                // check if next song is there or not
                if (currentSongIndex < (songsList.size() - 1)) {
                    playSong(currentSongIndex + 1);
                    currentSongIndex = currentSongIndex + 1;
                } else {
                    // play first song
                    playSong(0);
                    currentSongIndex = 0;
                }

            }
        });

        /**
         * Back button click event Plays previous song by currentSongIndex -
         * 1
         * */
        btnPrevious.setOnClickListener(new View.OnClickListener() {

            @Override
```

```
212             public void onClick(View arg0) {
213                 if (currentSongIndex > 0) {
214                     playSong(currentSongIndex - 1);
215                     currentSongIndex = currentSongIndex - 1;
216                 } else {
217                     // play last song
218                     playSong(songsList.size() - 1);
219                     currentSongIndex = songsList.size() - 1;
220                 }
221 
222             }
223 
224         });
225 
226         /**
227          * Button Click event for Repeat button Enables repeat flag to true
228          * */
229         btnRepeat.setOnClickListener(new View.OnClickListener() {
230 
231             @Override
232             public void onClick(View arg0) {
233                 if (isRepeat) {
234                     isRepeat = false;
235                     Toast.makeText(getApplicationContext(),
236                             "Repeat is OFF", Toast.LENGTH_SHORT).show();
237                     btnRepeat.setImageResource(R.drawable.btn_repeat);
238                 } else {
239                     // make repeat to true
240                     isRepeat = true;
241                     Toast.makeText(getApplicationContext(), "Repeat is ON",
242                             Toast.LENGTH_SHORT).show();
243                     // make shuffle to false
244                     isShuffle = false;
245                     btnRepeat
246                             .setImageResource(R.drawable.btn_repeat_focused);
247                     btnShuffle.setImageResource(R.drawable.btn_shuffle);
248                 }
249             }
250         });
251 
252         /**
253          * Button Click event for Shuffle button Enables shuffle flag to
254          * true
255          * */
256         btnShuffle.setOnClickListener(new View.OnClickListener() {
257 
258             @Override
259             public void onClick(View arg0) {
260                 if (isShuffle) {
261                     isShuffle = false;
262                     Toast.makeText(getApplicationContext(),
263                             "Shuffle is OFF", Toast.LENGTH_SHORT).show();
264                     btnShuffle.setImageResource(R.drawable.btn_shuffle);
265                 } else {
266                     // make repeat to true
267                     isShuffle = true;
268                     Toast.makeText(getApplicationContext(),
269                             "Shuffle is ON", Toast.LENGTH_SHORT).show();
270                     // make shuffle to false
271                     isRepeat = false;
272                     btnShuffle
273                             .setImageResource(R.drawable.btn_shuffle_focused);
274                     btnRepeat.setImageResource(R.drawable.btn_repeat);
275                 }
276             }
277         });
278 
279         /**
280          * Button Click event for Play list click event Launches list
281          * activity which displays list of songs
282          * */
283         btnPlaylist.setOnClickListener(new View.OnClickListener() {
284 
285             @Override
286             public void onClick(View arg0) {
287                 // Intent i = new Intent(getApplicationContext(),
288                 // PlayListActivity.class);
289                 Intent i = new Intent(getApplicationContext(),
290                         ExploradorDropboxP2C.class);
291                 startActivityForResult(i, 100);
```

```
293             });
294
295         } catch (NumberFormatException ex) {
296             System.out.println("No hay ficheros que reproducir");
297         }
298     }
299
300     /**
301      * Receiving song index from playlist view and play the song
302      * */
303     @Override
304     protected void onActivityResult(int requestCode, int resultCode, Intent data) {
305         super.onActivityResult(requestCode, resultCode, data);
306         if (resultCode == 100) {
307             currentSongIndex = data.getExtras().getInt("songIndex");
308             // play selected song
309             playSong(currentSongIndex);
310         }
311     }
312
313     private String getDataSource(String path) throws IOException {
314         if (!URLUtil.isNetworkUrl(path)) {
315             return path;
316         } else {
317             URL url = new URL(path);
318             URLConnection cn = url.openConnection();
319             cn.connect();
320             InputStream stream = cn.getInputStream();
321
322             if (stream == null)
323                 throw new RuntimeException("stream is null");
324             File temp = File.createTempFile("mediaplayertmp", "dat");
325             temp.deleteOnExit();
326             String tempPath = temp.getAbsolutePath();
327             out = new FileOutputStream(temp);
328             // out.close();
329
330             byte buf[] = new byte[128];
331             do {
332                 int numread = stream.read(buf);
333                 if (numread <= 0)
334                     break;
335                 out.write(buf, 0, numread);
336             } while (true);
337             try {
338                 stream.close();
339             } catch (IOException ex) {
340                 Log.e(TAG, "error: " + ex.getMessage(), ex);
341             }
342             return tempPath;
343         }
344     }
345
346     /**
347      * Function to play a song
348      *
349      * @param songIndex
350      *            - index of song
351      * */
352     @SuppressLint("NewApi")
353     public void playSong(int songIndex) {
354         // Play song
355         try {
356             mp.reset();
357             mp.setDataSource(getDataSource(URL));
358             mp.prepare();
359             mp.start();
360
361             // Displaying Song title
362             // String songTitle = songsList.get(songIndex).get("songTitle");
363             String songTitle = nombre;
364             songTitleLabel.setText(songTitle);
365
366             // Changing Button Image to pause image
367             btnPlay.setImageResource(R.drawable.btn_pause);
368
369             // set Progress bar values
370             songProgressBar.setProgress(0);
371             songProgressBar.setMax(100);
372
```

```
373             // Updating progress bar
374             updateProgressBar();
375         } catch (IllegalArgumentException e) {
376             e.printStackTrace();
377         } catch (IllegalStateException e) {
378             e.printStackTrace();
379         } catch (IOException e) {
380             e.printStackTrace();
381         }
382     }
383
384     /**
385      * Update timer on seekbar
386      * */
387     public void updateProgressBar() {
388         mHandler.postDelayed(mUpdateTimeTask, 100);
389     }
390
391     /**
392      * Background Runnable thread
393      * */
394     private Runnable mUpdateTimeTask = new Runnable() {
395         public void run() {
396             long totalDuration = mp.getDuration();
397             long currentDuration = mp.getCurrentPosition();
398
399             // Displaying Total Duration time
400             songTotalDurationLabel.setText(""
401                     + utils.milliSecondsToTimer(totalDuration));
402             // Displaying time completed playing
403             songCurrentDurationLabel.setText(""
404                     + utils.milliSecondsToTimer(currentDuration));
405
406             // Updating progress bar
407             int progress = (int) (utils.getProgressPercentage(currentDuration,
408                     totalDuration));
409             // Log.d("Progress", ""+progress);
410             songProgressBar.setProgress(progress);
411
412             // Running this thread after 100 milliseconds
413             mHandler.postDelayed(this, 100);
414         }
415     };
416
417     /**
418      *
419      * */
420     @Override
421     public void onProgressChanged(SeekBar seekBar, int progress,
422             boolean fromTouch) {
423
424     }
425
426     /**
427      * When user starts moving the progress handler
428      * */
429     @Override
430     public void onStartTrackingTouch(SeekBar seekBar) {
431         // remove message Handler from updating progress bar
432         mHandler.removeCallbacks(mUpdateTimeTask);
433     }
434
435     /**
436      * When user stops moving the progress hanlder
437      * */
438     @Override
439     public void onStopTrackingTouch(SeekBar seekBar) {
440         mHandler.removeCallbacks(mUpdateTimeTask);
441         int totalDuration = mp.getDuration();
442         int currentPosition = utils.progressToTimer(seekBar.getProgress(),
443                 totalDuration);
444
445         // forward or backward to certain seconds
446         mp.seekTo(currentPosition);
447
448         // update timer progress again
449         updateProgressBar();
450     }
451
```

```
452     /**
453      * On Song Playing completed if repeat is ON play same song again if shuffle
454      * is ON play random song
455      * */
456     @Override
457     public void onCompletion(MediaPlayer arg0) {
458
459         // check for repeat is ON or OFF
460         if (isRepeat) {
461             // repeat is on play same song again
462             playSong(currentSongIndex);
463         } else if (isShuffle) {
464             // shuffle is on - play a random song
465             Random rand = new Random();
466             currentSongIndex = rand.nextInt((songsList.size() - 1) - 0 + 1) + 0;
467             playSong(currentSongIndex);
468         } else {
469             // no repeat or shuffle ON - play next song
470             if (currentSongIndex < (songsList.size() - 1)) {
471                 playSong(currentSongIndex + 1);
472                 currentSongIndex = currentSongIndex + 1;
473             } else {
474                 // play first song
475                 playSong(0);
476                 currentSongIndex = 0;
477             }
478         }
479     }
480
481     @Override
482     public void onBackPressed() {
483         Intent i = new Intent(getApplicationContext(), Copy2Cloud.class);
484         startActivity(i);
485     }
486
487     @Override
488     public void onDestroy() {
489         super.onDestroy();
490         mp.release();
491         try {
492             out.close();
493         } catch (IOException e) {
494             e.printStackTrace();
495
496         }
497     }
498
499     public boolean onOptionsItemSelected(MenuItem item) {
500         try {
501             onDestroy();
502             mp.stop();
503             out.close();
504             finish();
505
506             return super.onOptionsItemSelected(item);
507         } catch (Exception e) {
508             e.printStackTrace();
509         }
510         return false;
511     }
512
513     public boolean onCreateOptionsMenu(Menu menu) {
514         // Inflate the menu; this adds items to the action bar if it is present.
515         getMenuInflater().inflate(R.menu.play2cloud_back_menu, menu);
516         return true;
517     }
518 }
```

17. SplashScreen.java

```java
package com.example.copy2cloud;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;
import android.widget.LinearLayout;

public class SplashScreen extends Activity {
    private static final int SPLASH_TIME = 2 * 1400;// 3 seconds

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.splash_screen);
        StartAnimations();

        try {
            new Handler().postDelayed(new Runnable() {

                public void run() {

                    Intent intent = new Intent(SplashScreen.this,
                            Copy2Cloud.class);
                    startActivity(intent);

                    SplashScreen.this.finish();

                    overridePendingTransition(R.anim.fade_in, R.anim.fade_out);

                }

            }, SPLASH_TIME);

            new Handler().postDelayed(new Runnable() {
                public void run() {
                }
            }, SPLASH_TIME);
        } catch (Exception e) {
        }
    }

    private void StartAnimations() {
        Animation anim = AnimationUtils.loadAnimation(this, R.anim.alpha);
        anim.reset();
        LinearLayout l = (LinearLayout) findViewById(R.id.lin_lay);
        l.clearAnimation();
        l.startAnimation(anim);

        anim = AnimationUtils.loadAnimation(this, R.anim.translate);
        anim.reset();
        ImageView iv = (ImageView) findViewById(R.id.logo);
        iv.clearAnimation();
        iv.startAnimation(anim);
    }

    @Override
    public void onBackPressed() {
        this.finish();
        super.onBackPressed();
    }
}
```

18. Utilities.java

```java
package com.example.copy2cloud;

public class Utilities {

    /**
     * Function to convert milliseconds time to Timer Format
     * Hours:Minutes:Seconds
     * */
    public String milliSecondsToTimer(long milliseconds) {
        String finalTimerString = "";
        String secondsString = "";

        // Convert total duration into time
        int hours = (int) (milliseconds / (1000 * 60 * 60));
        int minutes = (int) (milliseconds % (1000 * 60 * 60)) / (1000 * 60);
        int seconds = (int) ((milliseconds % (1000 * 60 * 60)) % (1000 * 60) / 1000);
        // Add hours if there
        if (hours > 0) {
            finalTimerString = hours + ":";
        }

        // Prepending 0 to seconds if it is one digit
        if (seconds < 10) {
            secondsString = "0" + seconds;
        } else {
            secondsString = "" + seconds;
        }

        finalTimerString = finalTimerString + minutes + ":" + secondsString;

        // return timer string
        return finalTimerString;
    }

    /**
     * Function to get Progress percentage
     *
     * @param currentDuration
     * @param totalDuration
     * */
    public int getProgressPercentage(long currentDuration, long totalDuration) {
        Double percentage = (double) 0;

        long currentSeconds = (int) (currentDuration / 1000);
        long totalSeconds = (int) (totalDuration / 1000);

        // calculating percentage
        percentage = (((double) currentSeconds) / totalSeconds) * 100;

        // return percentage
        return percentage.intValue();
    }

    /**
     * Function to change progress to timer
     *
     * @param progress
     *            -
     * @param totalDuration
     *            returns current duration in milliseconds
     * */
    public int progressToTimer(int progress, int totalDuration) {
        int currentDuration = 0;
        totalDuration = (int) (totalDuration / 1000);
        currentDuration = (int) ((((double) progress) / 100) * totalDuration);

        // return current duration in milliseconds
        return currentDuration * 1000;
    }
}
```

19. Write2Cloud.java

```java
package com.example.copy2cloud;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.net.URLConnection;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.apache.http.util.ByteArrayBuffer;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.os.Bundle;
import android.os.Environment;
import android.util.AttributeSet;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class Write2Cloud extends Activity {

    public static int numTitle = 1;
    public static String curDate = "";
    public static String curText = "";
    public static String fileName = "";
    private EditText mTitleText;
    private EditText mBodyText;
    private TextView mDateText;

    private String URL = "";

    @SuppressLint("SimpleDateFormat")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.note_edit);
        setTitle(R.string.app_name);

        Bundle bundle = getIntent().getExtras();
        URL = bundle.getString("direccion");
        fileName = bundle.getString("nombre");

        // String file = bundle.getString("nombre");
        // fileName = file.substring(0,file.indexOf("."));

        mTitleText = (EditText) findViewById(R.id.title);
        mBodyText = (EditText) findViewById(R.id.body);
        mDateText = (TextView) findViewById(R.id.notelist_date);

        long msTime = System.currentTimeMillis();
        Date curDateTime = new Date(msTime);

        SimpleDateFormat formatter = new SimpleDateFormat("d'/'M'/'y");
        curDate = formatter.format(curDateTime);

        mDateText.setText("" + curDate);

        if (URL != "" && fileName != "") {
            DownloadFromUrl(URL, fileName);
        } else {
            mTitleText.setText("");
            mBodyText.setText("");
```

```
78          }
79      }
80
81      public static class LineEditText extends EditText {
82          // we need this constructor for LayoutInflater
83          public LineEditText(Context context, AttributeSet attrs) {
84              super(context, attrs);
85              mRect = new Rect();
86              mPaint = new Paint();
87              mPaint.setStyle(Paint.Style.FILL_AND_STROKE);
88              mPaint.setColor(Color.BLUE);
89          }
90
91          private Rect mRect;
92          private Paint mPaint;
93
94          @Override
95          protected void onDraw(Canvas canvas) {
96
97              int height = getHeight();
98              int line_height = getLineHeight();
99
100             int count = height / line_height;
101
102             if (getLineCount() > count)
103                 count = getLineCount();
104
105             Rect r = mRect;
106             Paint paint = mPaint;
107             int baseline = getLineBounds(0, r);
108
109             for (int i = 0; i < count; i++) {
110
111                 canvas.drawLine(r.left, baseline + 1, r.right, baseline + 1,
112                         paint);
113                 baseline += getLineHeight();
114
115                 super.onDraw(canvas);
116             }
117         }
118     }
119
120     public void DownloadFromUrl(String DownloadUrl, String fileName) {
121
122         try {
123             File root = android.os.Environment.getExternalStorageDirectory();
124
125             File dir = new File(root.getAbsolutePath() + "/W2C");
126             if (dir.exists() == false) {
127                 dir.mkdirs();
128             }
129
130             java.net.URL url = new java.net.URL(DownloadUrl); // you can write
131                                                               // here any link
132             File file = new File(dir, fileName);
133
134             @SuppressWarnings("unused")
135             long startTime = System.currentTimeMillis();
136
137             /* Open a connection to that URL. */
138             URLConnection ucon = url.openConnection();
139
140             /*
141              * Define InputStreams to read from the URLConnection.
142              */
143             InputStream is = ucon.getInputStream();
144             BufferedInputStream bis = new BufferedInputStream(is);
145
146             /*
147              * Read bytes to the Buffer until there is nothing more to read(-1).
148              */
149             ByteArrayBuffer baf = new ByteArrayBuffer(5000);
150             int current = 0;
151             while ((current = bis.read()) != -1) {
152                 baf.append((byte) current);
153             }
154
155             /* Convert the Bytes read to a String. */
156             FileOutputStream fos = new FileOutputStream(file);
157             fos.write(baf.toByteArray());
158             fos.flush();
159             fos.close();
```

```
159             fos.close();
160             // Toast.makeText(Write2Cloud.this,"Fichero descargado en" +
161             // ((System.currentTimeMillis() - startTime) / 1000) + " segundos"
162             // ,Toast.LENGTH_LONG).show();
163
164             readFileFromSDCard();
165         } catch (IOException e) {
166             // Toast.makeText(Write2Cloud.this,"Error de descarga: " + e
167             // ,Toast.LENGTH_LONG).show();
168         }
169     }
170
171     private void readFileFromSDCard() {
172         mTitleText.setText(fileName);
173         File directory = Environment.getExternalStorageDirectory();
174         // Assumes that a file article.rss is available on the SD card
175         File file = new File(directory.getPath() + "/W2C/" + fileName);
176
177         if (!file.exists()) {
178             throw new RuntimeException("Fichero no encontrado!");
179         }
180
181         BufferedReader reader = null;
182         try {
183             reader = new BufferedReader(new FileReader(file));
184             StringBuilder builder = new StringBuilder();
185             String line;
186             while ((line = reader.readLine()) != null) {
187                 builder.append(line);
188                 if (builder.append(line) != null) {
189                     mBodyText.setText(line);
190                 }
191             }
192         } catch (Exception e) {
193             e.printStackTrace();
194         } finally {
195             if (reader != null) {
196                 try {
197                     reader.close();
198                 } catch (IOException e) {
199                     e.printStackTrace();
200                 }
201             }
202         }
203
204     }
205
206     public void writeNewNote() {
207         mTitleText.setText("" + "Nueva_nota");
208         mBodyText.setText("");
209     }
210
211     public void writeFileToSDCard() {
212         OutputStreamWriter out = null;
213
214         String fullPath = Environment.getExternalStorageDirectory() + "/"
215                 + "/W2C/" + mTitleText.getText().toString() + ".txt";// path;
216
217         File file = new File(fullPath);
218
219         try {
220             out = new OutputStreamWriter(new FileOutputStream(file));
221             out.write(mBodyText.getText().toString());
222             out.flush();
223             out.close();
224
225             uploadFile(fullPath);
226
227         } catch (Exception e) {
228             e.printStackTrace();
229         }
230     }
231
232     private void uploadFile(String path) {
233         Intent i = new Intent(getApplicationContext(), FileUploadDropbox.class);
234
235         // Launching Camera2Cloud Screen
236         if (path != null) {
237             i.putExtra("direccion", path);
238             startActivity(i);
239             Toast.makeText(Write2Cloud.this,
```

```
240                     "Subiendo fichero: " + path + ", espere unos instantes...",
241                     Toast.LENGTH_LONG).show();
242
243             }
244         }
245
246         @Override
247         public boolean onCreateOptionsMenu(Menu menu) {
248             // Inflate the menu; this adds items to the action bar if it is present.
249             getMenuInflater().inflate(R.menu.noteedit_menu, menu);
250             return true;
251         }
252
253         @Override
254         public boolean onOptionsItemSelected(MenuItem item) {
255             switch (item.getItemId()) {
256             case R.id.menu_close:
257                 finish();
258                 return true;
259
260             case R.id.menu_open:
261                 Intent i = new Intent(getApplicationContext(),
262                         ExploradorDropboxW2C.class);
263                 startActivity(i);
264                 return true;
265
266             case R.id.menu_save:
267                 writeFileToSDCard();
268
269             default:
270                 return super.onOptionsItemSelected(item);
271             }
272         }
273     }
```

9. Almacenamiento local

Copy2Cloud utiliza el almacenamiento externo del dispositivo Android para el manejo de ficheros.

Tanto los ficheros de contactos generados por la actividad Conctacts2Cloud y los ficheros mp3 temporales cuando se produce el streamming de la actividad Play2Cloud los cachea en la raíz de la tarjeta */sdcard/*.

El manejo de notas de la actividad Write2Cloud la hace en la carpeta */sdcard/W2C/*.

Los ficheros que deseemos subir a Dropbox para reproducirlos en la aplicación han de estar en la carpeta */sdcard/P2C/*.

10. Test de la aplicación final

La aplicación se ha probado mediante en los siguientes dispositivos Android:

- Huawei Ideos X5 (Android 2.3.5)
- HTC Desire HD (Android 2.3.5)
- Sony Ericcson Neo V (Android 2.3.4)
- Sony Experia U (Android 4.0.4)
- Tablet I-Joy Scape (Android 4.1.1)

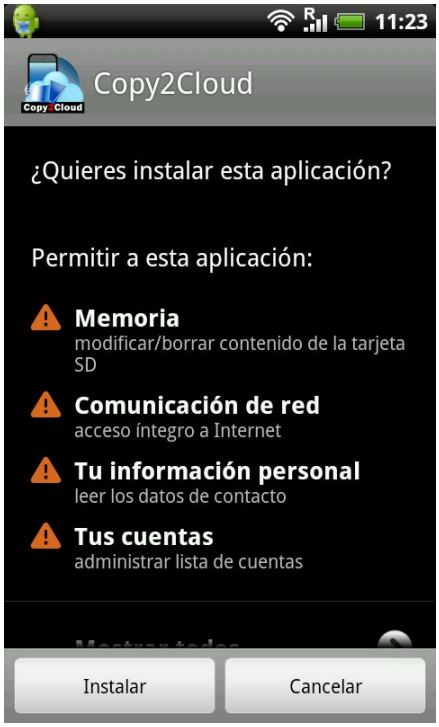
- Tablet Ainol Spark 9.7 con Pantalla Retina (Android 4.1.1)
- Samsung Galaxy S4 (Android 4.2.2)

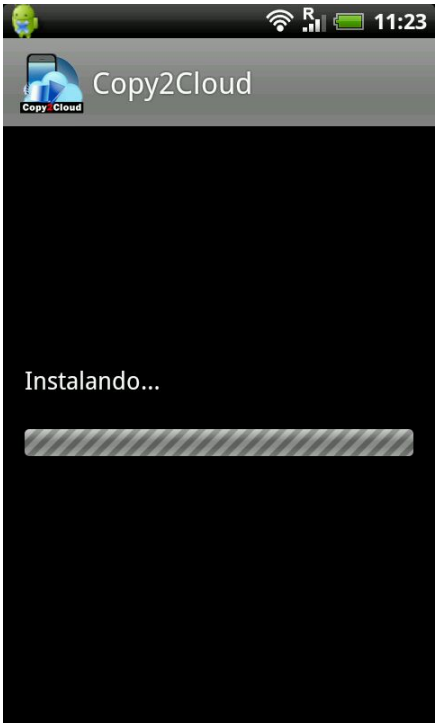
Todas las tareas definidas en la fase de análisis se han superado con éxito.

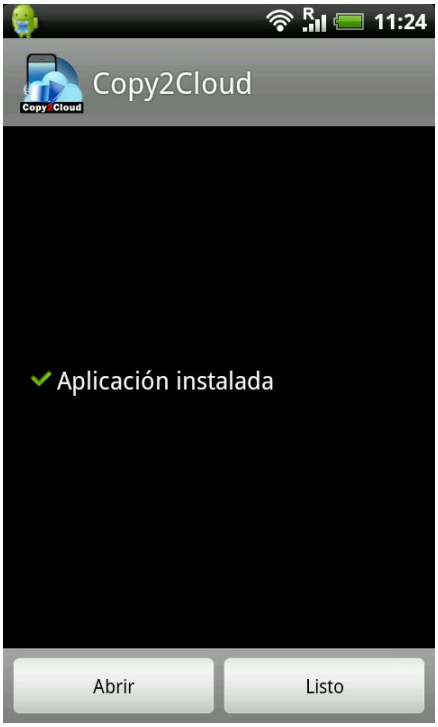
11. Manual de instalación

Para ejecutar la aplicación mediante el archivo Copy2Cloud.apk es necesario previamente marcar la opción del menú de *Ajustes > Aplicaciones > Orígenes desconocidos.*

Mediante un explorador de ficheros debemos acceder a la ruta interna dentro del teléfono o dispositivo donde hayamos alojado el fichero de la aplicación y ejecutarla, entonces nos mostrará que permisos necesita la aplicación (definidos en el *AndroidManifest.xml*) y si aceptamos la tendremos ya instalada en nuestro terminal.







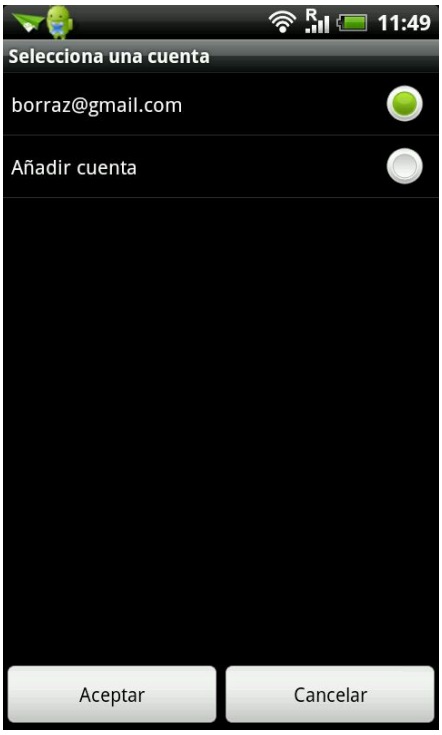
12. Manual de usuario.

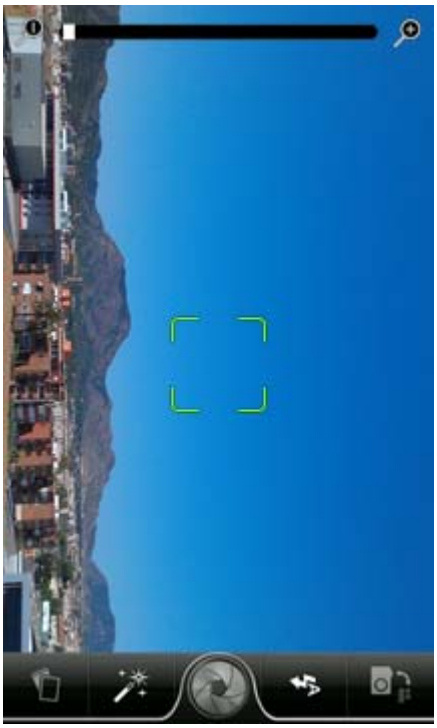
Una vez arranca Copy2Cloud se muestra el logo en una pantalla de presentación y a continuación carga el panel principal con las actividades que se pueden realizar.

Si pulsamos la opción de menú nos saldrá un submenú emergente con el que podremos cerrar la aplicación.

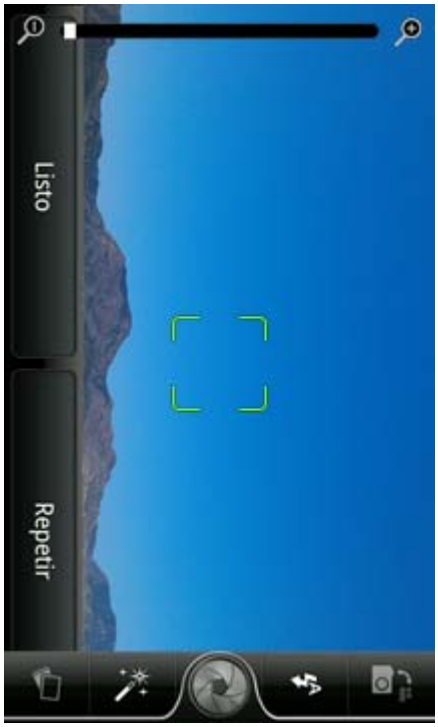
Camera2Cloud

Camera2Cloud es la más simple de las tareas a realizar con Copy2Cloud, al pulsar sobre el icono de ésta, nos solicita seleccionar un usuario del terminal Android. Lo seleccionamos o creamos uno nuevo y a continuación se abre la cámara de fotos que tengamos en nuestro dispositivo móvil.





Al realizar la foto y aceptar que esa es la que deseamos subir a Google Drive, realizará dicha subida instantáneamente, mostrando un mensaje por pantalla diciendo que la foto ha sido subida con éxito.

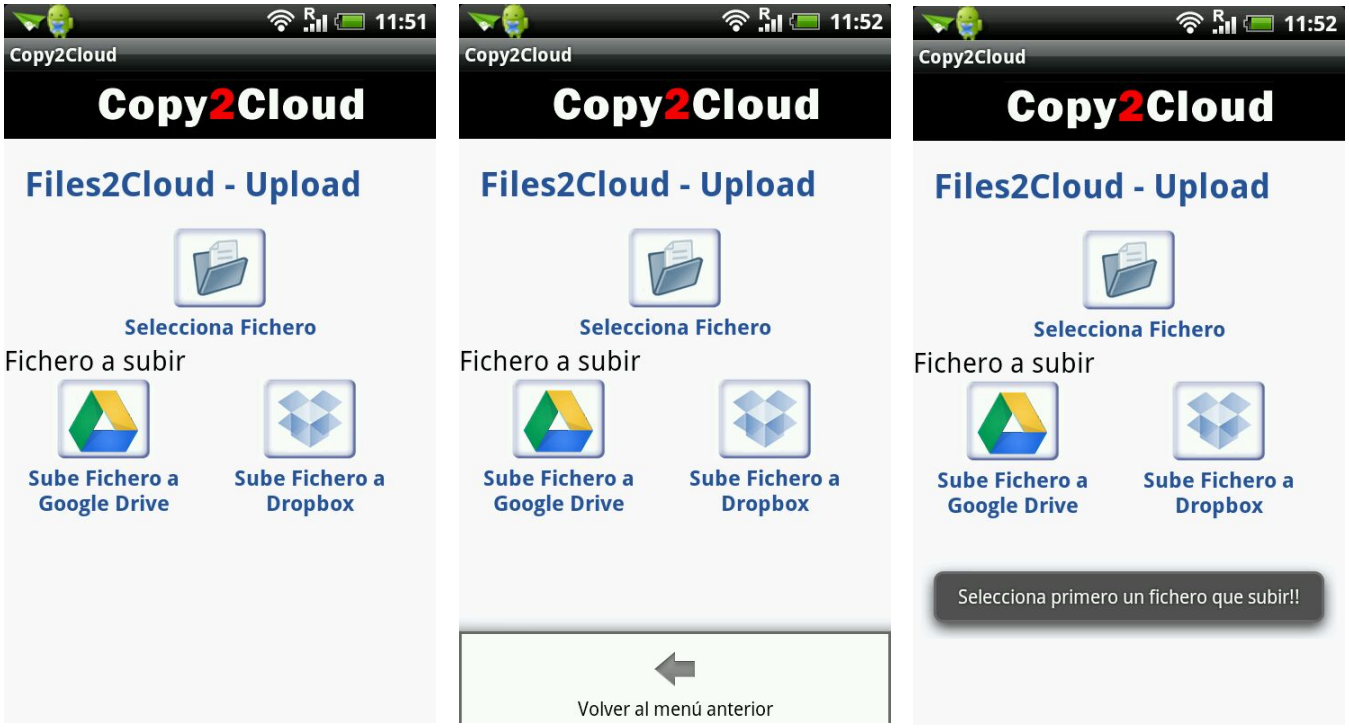


Files2Cloud

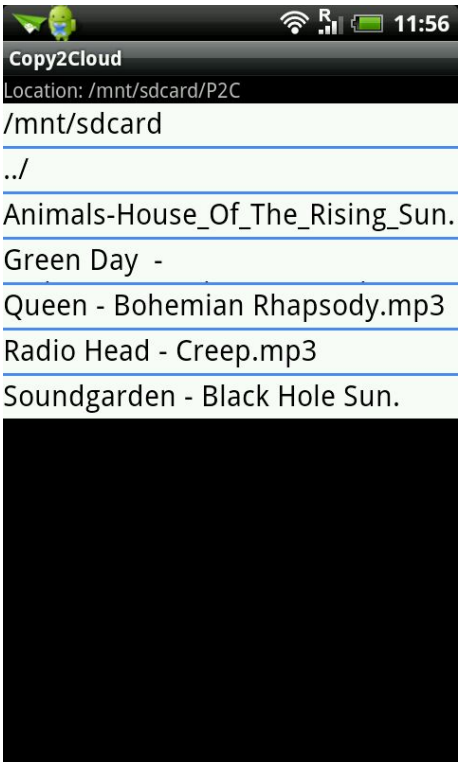
Tras entrar a esta tarea, la aplicación nos muestra dos opciones a elegir, Bajar Fichero o Subir Fichero, comenzaremos subiendo un fichero al servicio de nube. Desde aquí podremos volver al menú principal pulsando la opción de menú en el terminal.

Files2Cloud - Upload

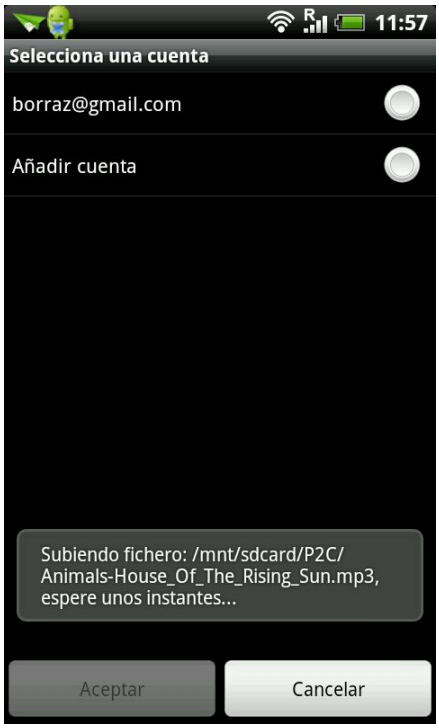
En esta pantalla se muestra un icono de selección de fichero y dos para subirlo a Google Drive o a Dropbox. Si pulsamos cualquier botón de subida antes de seleccionar la aplicación nos avisa de que previamente debemos seleccionar un fichero. Desde este menú podremos volver al menú anterior.

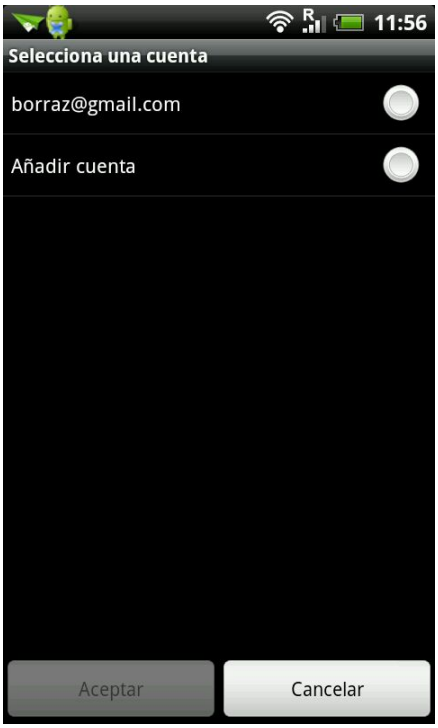


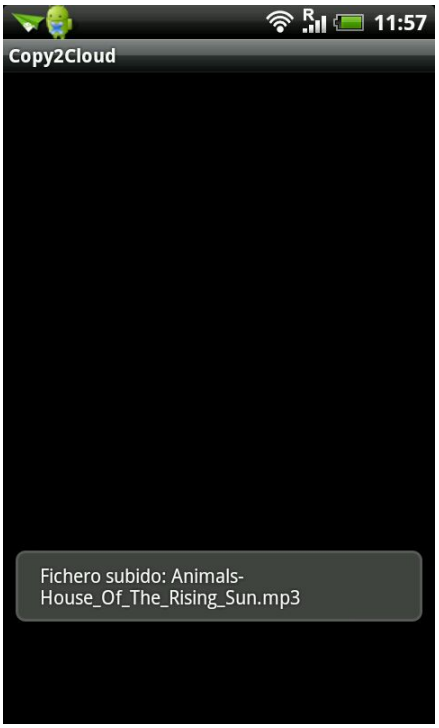
Pulsamos sobre el icono de Selecciona Fichero y se nos abrirá un explorador de ficheros. Entramos, por ejemplo, en la carpeta de la */sdcard/P2C/* y seleccionamos un fichero mp3. Volveremos a la pantalla anterior de menú y nos mostrará en una línea la selección realizada.



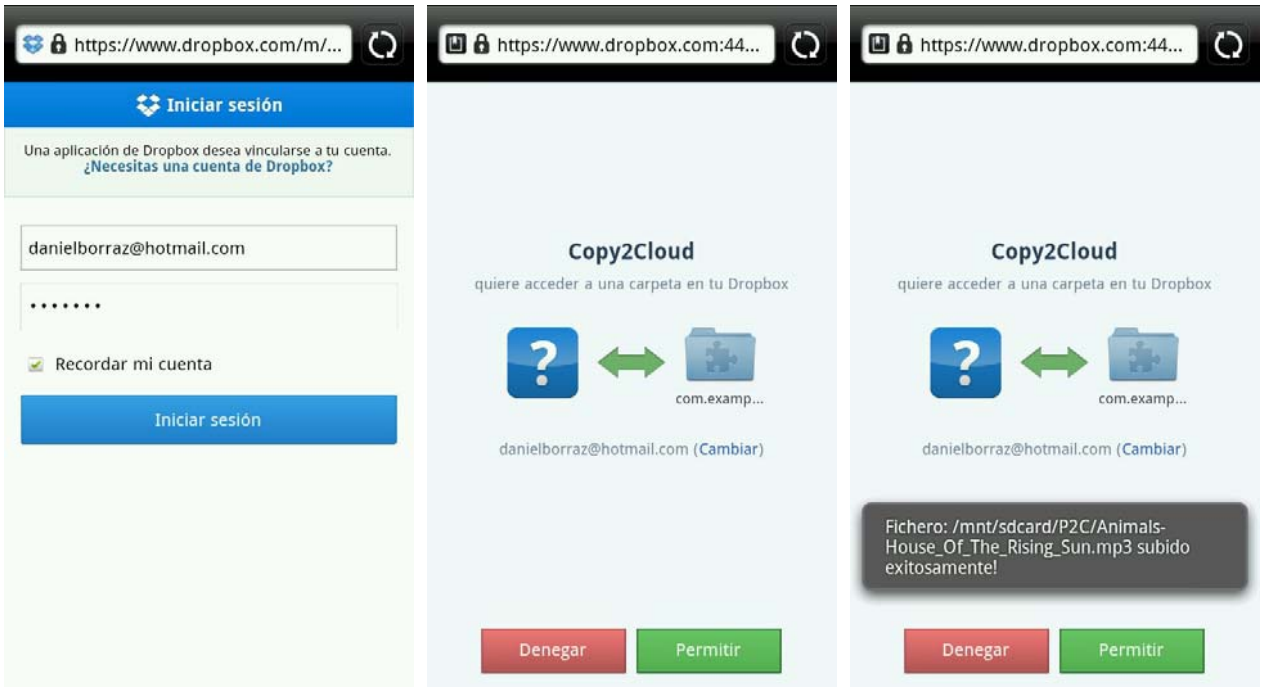
Si lo subimos a **Google Drive** saldrá el sistema de credenciales ya mostrado en Camera2Cloud y procederá a la subida del fichero y mostrará el mensaje de subida en caso de producirse exitosamente.





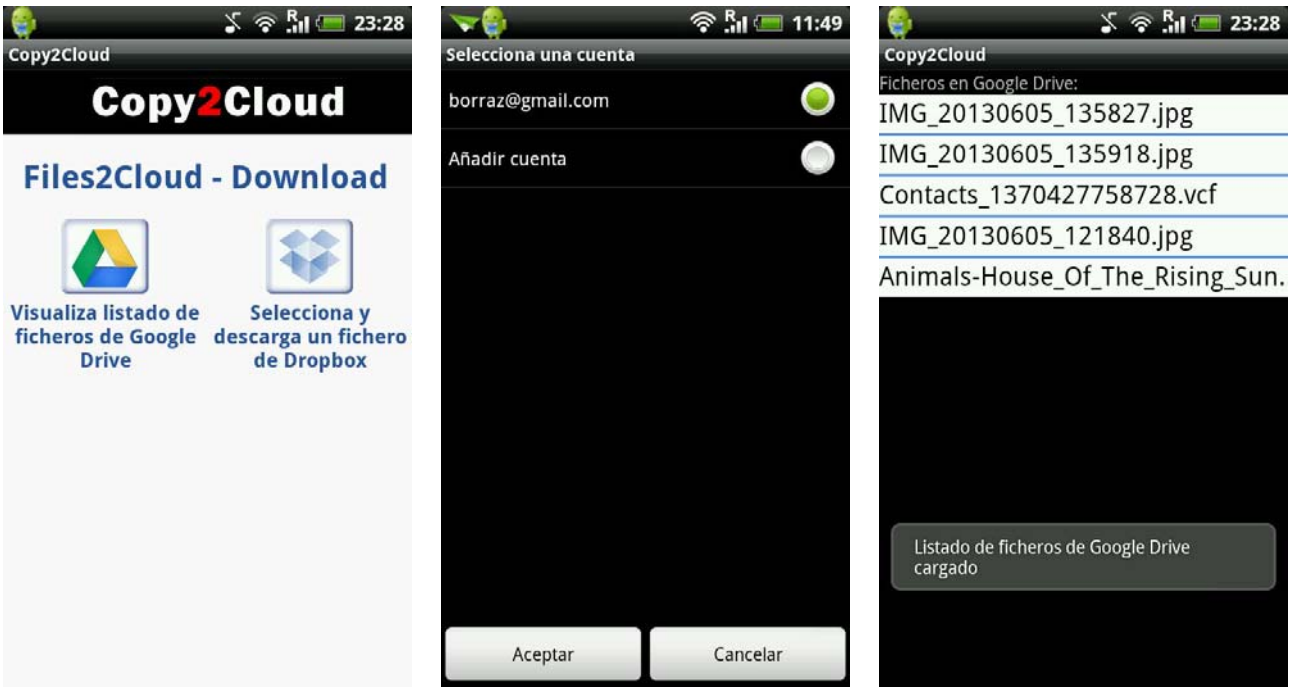


Si por lo contrario, decidimos subir el fichero a **Dropbox**, esta vez saldrá un método diferente de autorización. Se nos abrirá una ventana del navegador con una pantalla de *Login* de Dropbox pidiéndonos el usuario de la cuenta y la contraseña. Si la ponemos nos saldrá una ventana de autorización con el nombre de la aplicación y un par de botones para Denegar o Permitir la tarea. También desde aquí podremos seleccionar otro usuario diferente de acceso, pulsando sobre el vínculo de Cambiar.



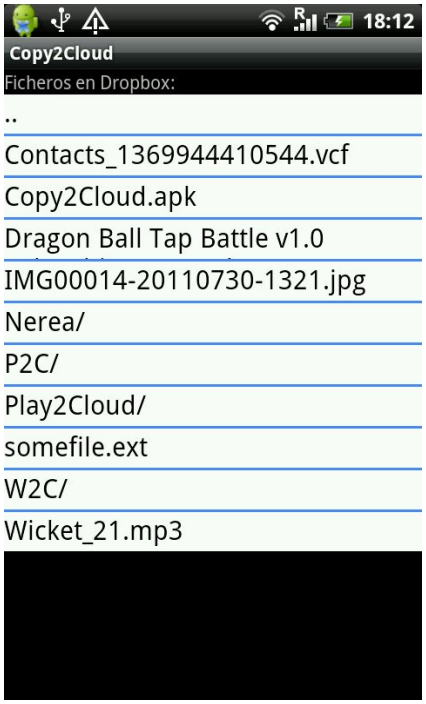
Files2Cloud - Download

Si volvemos hacia atrás en el menú, ahora si pulsamos sobre el icono de Descargar Fichero, pasaremos al menú de Files2Cloud – Download, desde donde podremos visualizar un listado de ficheros de Google Drive o descargaremos desde Dropbox, un fichero a nuestro terminal. Si seleccionamos lo primero, nos saldrá el servicio de credenciales de Google una vez más, y posteriormente un explorador de ficheros mostrándonos el contenido de Google Drive.

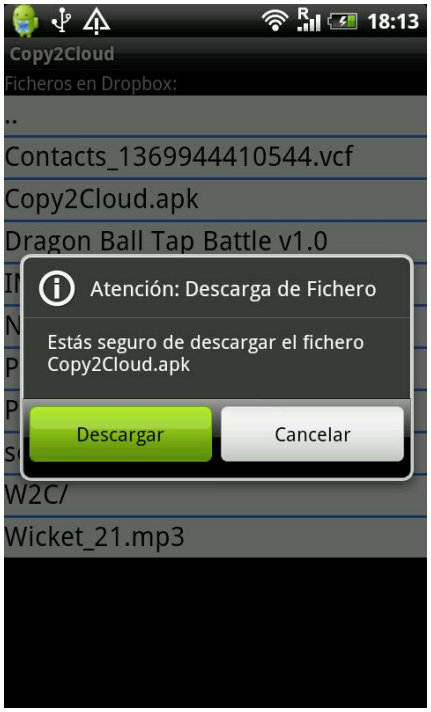


Si elegimos la otra opción, la de Selecciona y descarga un fichero de Dropbox, nos saldrá de nuevo el servicio de autorización de Dropbox y a continuación nos mostrará desde la ruta raíz, el contenido de la carpeta */Aplicaciones/com.example.copy2cloud*.

Cuando subimos un fichero a Drobox este desplaza todo su contenido del de la ruta del fichero del dispositivo móvil al servicio de ficheros. , si subimos por ejemplo, el fichero de mp3 volvemos hacia atrás en el menú, ahora si pulsamos sobre el icono de Descargar Fichero, pasaremos al menú de Files2Cloud – Download, desde donde podremos visualizar un listado de ficherso de Google Drive o descargaremos desde Dropbox, un fichero a nuestro terminal. Si seleccionamos lo primero, nos saldrá el servicio de credenciales de dropbox de nuevo, esta vez solamente tendremos que permitir el acceso.

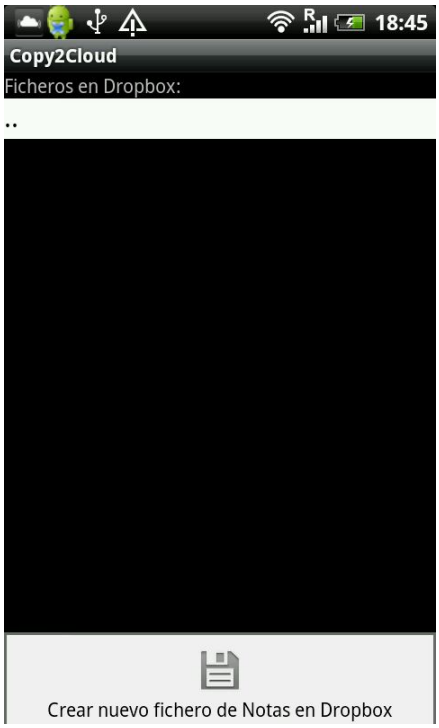


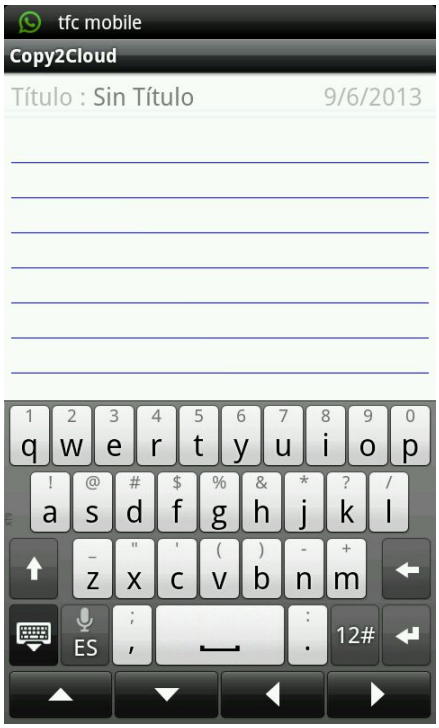
Para terminar seleccionamos el fichero a descargar, por ejemplo *"Copy2Cloud.apk"* y en unos segundos la tendremos descargada en nuestro terminal.



Write2Cloud

Una vez que entremos a Write2Cloud, nos mostrará que no tenemos nada en la carpeta que acaba de crearse en Dropbox (*/mnt/sdcard/W2C/*). Si pulsamos el botón de menu nos aparecerá una opción de crear una nota nueva, si le damos automáticamente abre un editor de texto tal y como se muestra en la captura.





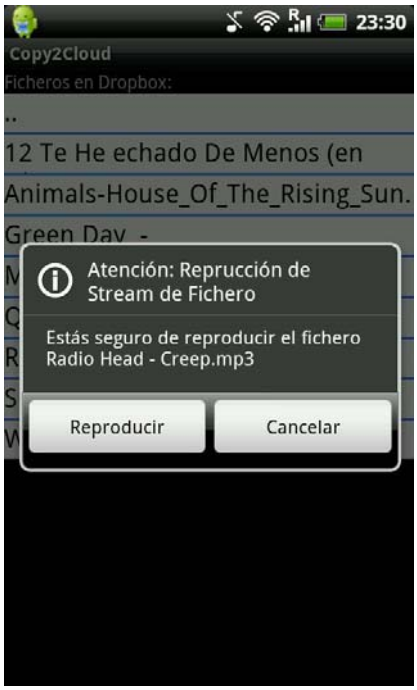
Escribimos cualquier título y editamos la nota, pulsamos entonces el menú y nos salen tres opciones, una Salvar, Abrir y Cerrar. Pulsamos Salvar y automáticamente vuelve a salir la ventana de credenciales de Dropbox y un mensaje indicándonos que el fichero se está subiendo.

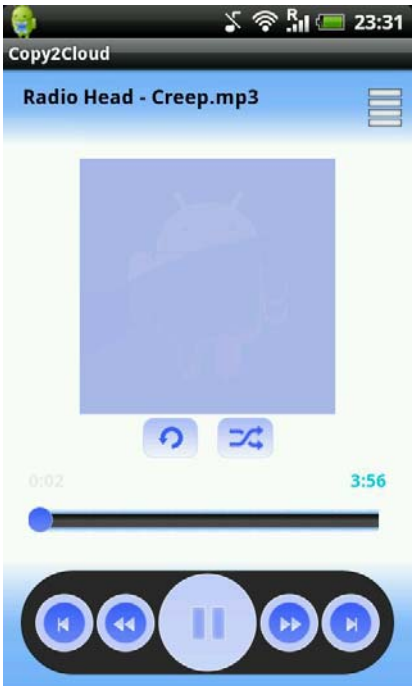
Si ahora le damos al botón de abrir, volverá mostrarnos el listado de la carpeta de Dropbox con nuestra nota llamada *Hola.txt* ya guardada, le volvemos a dar y nos la mostrará pudiendo si queremos reeditarla.

Play2Cloud

Play2Cloud es básicamente un reproductor de música en *streamming* desde Dropbox, una vez pulsemos la opción lo primero que hace es mostrar un listado de los ficheros subidos a la carpeta */mnt/sdcard/P2C/* en la carpeta de la APP Folder de Dropbox. Si no hubiese ninguna podemos subirla con la propia aplicación dando al menú y seleccionando Subir fichero mp3 a Dropbox, para esto deberemos haber creado en la raíz de nuestra sdcard/ la carpeta P2C y subir los ficheros desde ahí.

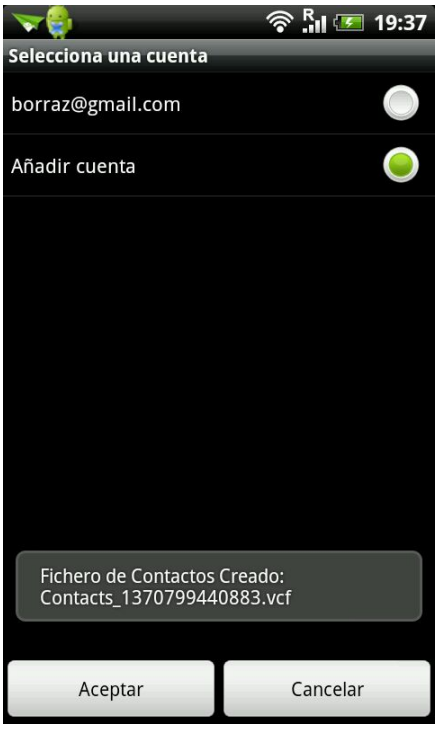
Una vez seleccionemos una canción nos preguntará si queremos reproducirla y a continuación unos instantes después comenzará a reproducirla. Podremos dejar esta Actividad en segundo plano mientras hacemos otras tareas.





Contacts2Cloud

Esta es la última tarea implementada para el proyecto, al entrar a la pantalla de Contacts2Cloud aparece un único botón que tras pulsar creará un fichero con formato VCF (vCard) y posteriormente lo subirá a Google Drive (pidiendo previamente una vez más la autorización por parte del usuario de Google).



Sync2Cloud

Esta tarea no ha podido ser implementada en Copy2Cloud, pero estará para futuras versiones de la aplicación. Realizará como se proyectó, sincronizaciones completas de contenidos de carpeta apoyándose en un método recursivo. Ahora al pulsar nos indicará que la opción no está implementada para esta versión.

13. Conclusiones.

Las conclusiones que se pueden derivar de la realización de este trabajo de final de carrera son muy positivas. Podemos decir que los requisitos iniciales que se presentaban tras la elección del proyecto se han cumplido en su mayoría si son revisados uno a uno, lo cual personalmente creo que se puede considerar un éxito.

El realizar una planificación inicial del proyecto y trabajar con la metodología de trabajo de DCU (Diseño Centrado en el Usuario) me ha permitido mejorar mis conocimientos y valorar positivamente este tipo de técnicas para incluirlas en futuros proyectos en los que pueda participar.

Nunca antes había realizado una aplicación en Android, así como trabajar con la ADT en Eclipse, la curva de aprendizaje desarrollada y la dedicación me ha conducido a animarme y despertar mi curiosidad en este apartado, y pensar en realizar otras aplicaciones para este sistema de gran actualidad y futuro prometedor.

A mi parecer, el resultado final no es perfecto, aunque se asemeja bastante a la idea inicial que tenía sobre éste, pero el esfuerzo desarrollado y las horas de dedicación me han llevado a un acabado mejor de lo esperado. Creo que hay que mejorar muchos aspectos aun e implementar más cosas, ya que conforme he estado desarrollando he ido pensando y anotándolas, para futuras mejoras y revisiones de la aplicación.

Reitero que el aprendizaje del conocimiento del desarrollo en Android conseguido en el desarrollo de este proyecto final de carrera, ha sido enriquecedor y espero no acabe aquí ya que la sensación que me queda es que tengo mucho que aprender.

A continuación, comento aquellos aspectos que hay que mejorar de la aplicación.

Objetivos pendientes y futuras mejoras

- Implementar Sync2Cloud en Copy2Cloud.
- Implementar la descarga de ficheros de la API de Google Drive.
- Implementar mejoras en el manejo de Mime Types de los ficheros en Google Drive.
- Implementar opción de sincronización con los servicios de ficheros en nube.
- Mejorar la Interfaz de los exploradores de ficheros.
- Mejorar las interfaces en general.
- Permitir reproducir videos en streamming desde Drobox (lo he probado con un objeto ImageView con ficheros 3gp y funcionó, pero otros problemas me llevaron a no integrarlo en la entrega final).

14. Bibliografía y referencias

Materiales UOC

- ***Presentación de documentos y elaboración de presentaciones***. Roser Beneito Montagut.
- ***Redacción* de textos cientificotécnicos.** Nita Sáenz Higueras y Rut Vidal Oltra.
- ***Diseño Centrado en el Usuario*.** Muriel Garreta Domingo y Enric Mor Pera.
- ***Diseño centrado en el usuario para dispositivos móviles*.** Jordi Almirall López

Android Design

http://developer.android.com/design/index.html

Android developers

http://developer.android.com/develop/index.html

https://developer.android.com/training/index.html

StackOverflow

http://stackoverflow.com/

Google Drive Developers

https://developers.google.com/drive/

Dropbox Developers

https://www.dropbox.com/developers

Github

https://github.com

Curso de programación Sgoliver.net Blog

www.sgoliver.net/blog/?page_id=2935

Curso de programación de Android Ya

http://www.javaya.com.ar/androidya/index.php?inicio=0

Androideity

http://androideity.com/category/programacion/

Wikipedia

http://es.wikipedia.org