APLICACIÓN ANDROID DE GESTIÓN DE STORAGE CLOUDS
UNISTOR

CARLOS ESTÉVEZ NOVO
MÁSTER EN INGENIERÍA WEB
PROYECTO DE FIN DE MÁSTER
Contenido

1 Introducción

1.1 Motivaciones

1.2 Alcance del proyecto

1.3 Estructura de la memoria

2 Servicios y tecnologías

2.1 Dropbox

2.1.1 Funcionamiento

2.1.2 Tipos de cuenta

2.1.3 Desarrollo

2.2 Box

2.2.1 Tipos de cuenta

2.2.2 Desarrollo

2.3 Android

2.3.1 Historia de Android

2.3.2 Principales características

2.3.3 Versiones de la plataforma

2.3.4 Arquitectura

2.4 Android Studio

2.4.1 Estructura de un proyecto

2.4.2 Gradle

3 Análisis Funcional

3.1 Características básicas

3.1.1 Gestión de cuentas

3.1.2 Operaciones básicas con ficheros

3.1.3 Subida y descarga de ficheros

3.2 Casos de uso

3.2.1 Especificación de casos de uso

4 Construcción

4.1 Estructura del proyecto Unistor

4.2 Diagrama de paquetes
4.3 Diagrama de clases ........................................................................................................ 40
  4.3.1 Paquete activity .................................................................................................... 40
  4.3.2 Paquete adapter .................................................................................................. 42
  4.3.3 Paquete asynctask .............................................................................................. 43
  4.3.4 Paquete útil .......................................................................................................... 44

4.4 Implementación ......................................................................................................... 46
  4.4.1 Descarga de ficheros .......................................................................................... 46
  4.4.2 Presentación de elementos de cuenta ................................................................. 52

5 Conclusiones .............................................................................................................. 58

Bibliografía ...................................................................................................................... 59
1 Introducción

1.1 Motivaciones

La vida de las personas tiende a volverse cada vez más centrada en el mundo digital. Gran parte de los procesos cotidianos que normalmente se realizaban de forma “presencial” o “real”, ahora se han trasladado al mundo electrónico. Nos mantenemos informados a través de diarios y revistas online. Realizamos vida social a través de plataformas online, las redes sociales. Trabajamos en entornos colaborativos online. Realizamos compras a través de plataformas online. Gestionamos nuestras finanzas y ahorros en sitios web. Por tanto, no sólo el plano personal de las vidas de las personas ha pasado a ser más digital, sino también otras facetas, como por ejemplo, la profesional. Tiene sentido que el almacenamiento de información también pase a estar en este entorno que, además de comodidad, aporta seguridad, rapidez e ubicuidad.

No sólo la existencia de Internet posibilita esto ya que, tradicionalmente, su uso estaba limitado por la utilización de los ordenadores, algo que, al fin y al cabo, fuerza su localización más o menos fija. Con el paso del tiempo, la proliferación de teléfonos inteligentes, smartphones, ha permitido tener en todas partes y en todo momento, acceso a la red de forma simple y directa. Esto, a su vez, ha facilitado que la vida de las personas se traslade, de nuevo, a este mundo digital.

Es la unión de estas dos razones el motivo principal de la realización de esta aplicación. Dado que gran parte de nuestra información (léase documentos, fotos, etc...) de toda índole está almacenada en formato digital y el que el uso de dispositivos inteligentes facilita el acceso a este mundo, su convergencia ha significado el impulso de servicios de almacenamiento online. Existen muchas alternativas, no siendo excluyentes, siendo muy habitual que una misma persona pueda usar varios de estos servicios, cada uno con sus características y enfoque. Esto genera una clara necesidad: una manera fácil y rápida de gestionar estos servicios, siendo deseable la posibilidad de hacerlo todo de la misma forma. Unistor cubre parcialmente esta necesidad, ya que es una aplicación para este tipo de terminales, los Smartphone Android, el sistema operativo predominante para estos terminales.
1.2 Alcance del proyecto

Los objetivos del proyecto serán los mismos que habían sido fijados en la propuesta de proyecto, presentada en su día. Fundamentalmente, la meta es la implementación de una aplicación sobre Android de gestión de diferentes servicios de nubes de almacenamiento. En particular, se pueden diferenciar los siguientes puntos:

- Desarrollo sobre Android, versión mínima 4.0.
- Utilización de API de distintos servicios de almacenamiento en la nube (Dropbox y Box, finalmente).
- Interfaz de gestión unificada.
- Operaciones con los ficheros almacenados.
  - Copiar.
  - Mover.
  - Descargar.
  - Subir fichero.
  - Eliminar fichero.

No forman parte del proyecto los siguientes puntos que, por ser similares y cercanos a los objetivos marcados, puedan parecer incluidos:

- Mover/copiar archivos/carpetas entre servicios de almacenamiento.
- Crear carpetas.
- Renombrar elementos
- Selección de destino de descarga de los ficheros.
1.3 Estructura de la memoria

El presente documento pretende ser una explicación del dominio de la aplicación y una pequeña muestra del diseño e implementación de la misma. Para ello, se ha organizado la memoria en 5 capítulos.

El capítulo 1: Introducción, detalla las razones que motivan un proyecto como este y realiza una breve introducción a la aplicación, determinando el alcance del mismo. También explica la organización de este documento.

El capítulo 2: Servicios y tecnologías, plantea una explicación de las características básicas de los servicios de almacenamiento utilizados, así como sus posibilidades abiertas a desarrolladores. Por otro lado, realiza una revisión del SO Android y su arquitectura. Para terminar, presenta el entorno de desarrollo con el que se hará el desarrollo de la aplicación.

El capítulo 3: Análisis Funcional, profundiza en el funcionamiento esperado de la aplicación, abundando en sus características más destacables, y dividiéndolas por casos de uso, que aportarán claridad sobre cada uno de los posibles usos de la aplicación.

El capítulo 4: Construcción, tratará, conjuntamente, el diseño y la implementación del software. En cuanto al diseño, se presentarán diagramas de paquetes con sus correspondientes relaciones entre ellos. Posteriormente, se mostrarán los diagramas de clases de esos paquetes, explicando la responsabilidad de cada una de las clases y sus relaciones. Por otra parte, la sección de la implementación muestra el código de las funcionalidades más relevantes del proyecto, con las explicaciones pertinentes para su correcto entendimiento.

El capítulo 5: Conclusiones, realizará una revisión de los objetivos iniciales del proyecto y valorará su cumplimiento. También planteará posibles futuras líneas de desarrollo.
2 Servicios y tecnologías

2.1 Dropbox

Dropbox es un servicio multiplataforma de alojamiento de archivos en la nube. Permite a los usuarios mantener una sincronización entre los archivos en línea y en el ordenador, enlazando todas las computadoras mediante una sola carpeta, lo cual constituye una manera fácil de respaldar y sincronizar los archivos. Además permite compartir carpetas y archivos con otros. Dispone de aplicación nativa para Windows, Linux, Mac OS X, Blackberry, Windows Phone, Android e iOS.

2.1.1 Funcionamiento

El servicio "cliente de Dropbox" permite a los usuarios dejar cualquier archivo en una carpeta designada. Ese archivo es sincronizado en la nube y en todas las demás computadoras del cliente de Dropbox. Los archivos en la carpeta de Dropbox pueden entonces ser compartidos con otros usuarios de Dropbox, ser accedidos desde la página Web de Dropbox o bien ser consultados desde el enlace de descarga directa que puede ser consultado tanto de la versión web como desde la ubicación original del archivo en cualquiera de los ordenadores en las que se encuentre. Asimismo, los usuarios pueden grabar archivos manualmente por medio de un navegador web.

Si bien Dropbox funciona como un servicio de almacenamiento, se enfoca en sincronizar y compartir archivos, y con un sistema que también permite hacerlo mediante usb. Además posee soporte para historial de revisiones, de forma que los archivos borrados de la carpeta de Dropbox pueden ser recuperados desde cualquiera de las computadoras sincronizadas. También existe la funcionalidad de conocer la historia de un archivo en el que se esté trabajando, permitiendo que una persona pueda editar y cargar los archivos sin peligro de que se puedan perder las versiones previas. El historial de los archivos está limitado a un periodo de 30 días, aunque existe una versión de pago que ofrece el historial ilimitado. El historial utiliza la tecnología de delta encoding. Para conservar ancho de banda y tiempo, si un archivo en una carpeta Dropbox de un usuario, es cambiado, Dropbox sólo carga las partes del archivo que son cambiadas cuando se sincroniza. Si bien el cliente de escritorio no tiene restricciones para el tamaño de los archivos, los archivos cargados por medio de la página Web están
limitados a un máximo de 300 MB cada uno. Dropbox utiliza el sistema de almacenamiento S3 de Amazon para guardar los archivos y SoftLayer Technologies para su infraestructura de apoyo, es decir, servidores, hosting y cloud computing.

### 2.1.2 Tipos de cuenta

El servicio de Dropbox se presta tanto a usuarios particulares como a empresas, contemplando diferentes planes de precios, que ofrecen diferentes condiciones de servicio:

- **Cuenta básica**: llamada Cuenta Dropbox, está pensada para uso personal y es gratuita, con un espacio de 2GB, ampliables mediante ciertas tareas que propone el propio servicio, como invitar amigos, enlazar la cuenta de determinadas redes sociales como Facebook o Twitter, etc…

- **Cuenta pro**: llamada Cuenta Dropbox Pro, pensada para uso personal pero para fines de mayor envergadura. Ofrece diferentes cantidades de espacio a contratar con diferentes precios cada uno, existiendo la posibilidad de 100GB por 9.99€/mes, 200GB por 19.99€/mes o 500GB por 49.99€/mes.

- **Cuenta empresa**: pensada para almacenar los documentos del trabajo de una empresa, ofrece amplias posibilidades de administración. Ofrece una cuenta individual para cada miembro del equipo de trabajo, siendo obligatorios 5 usuario como mínimo, dando un espacio de 1000GB para estos primeros 5. El precio con estas condiciones es de 795€/año. Cada usuario a mayores aumenta el precio a razón de 125€/año cada uno.

Más allá de las diferencias en cuanto a tamaño de cuenta, el servicio de Dropbox abarca características comunes a todas las cuentas, como el cifrado SSL y AES de 256 bits y la verificación en dos pasos y códigos móviles para el inicio de sesión.

En la siguiente tabla se presenta el detalle de las diferencias entre los tipos de cuentas ofertadas:
2.1.3 Desarrollo

Dropbox pone a disposición de los desarrolladores tres API’s diferentes, con finalidades diferentes, de forma que se simplifique el trabajo de los desarrolladores. La API básica es llamada Core API, y permite acceso completo al contenido de una cuenta Dropbox, incluyendo búsquedas, revisiones de ficheros, además de operaciones comunes como descarga y subida de ficheros. Sync API facilita el trabajo de edición de ficheros, liberando al programador de tener que preocuparse por tareas como la edición offline, cacheado de ficheros y la sincronización de cambios concurrentes. Por último,
Datastore API constituye una posibilidad para los programadores de almacenar datos estructurados en Dropbox que sean usados en sus aplicaciones a modo de bases de datos, encargándose esta API de tareas de sincronización y volcado. En el marco de este proyecto, la API más relevante es Core API, ya que la aplicación necesitará acceso total al contenido de la cuenta Dropbox.

En cuanto a esta API, Dropbox provee diferentes SDK’s para diferentes lenguajes de programación, abarcando Python, Ruby, PHP, Java, Android, iOS y Mac OS X. Sin el uso de SDK, existe la posibilidad de usar la API REST directamente sobre HTTP.

2.2 Box

Es un servicio web de almacenamiento e intercambio de archivos en la nube, inicialmente lanzado para el mundo empresarial. Existe aplicación nativa para Windows, Mac OS X, Android, Windows Phone, iOS, BlackBerry y WebOS. El núcleo del servicio se basa en compartir, colaborar y trabajar con archivos que se suben a Box. Box ofrece 3 tipos de cuenta: empresariales, comerciales y personales. En función del tipo de cuenta, box tiene características tales como capacidad de almacenamiento ilimitada, personalización de marca y los controles administrativos.

2.2.1 Tipos de cuenta

Actualmente, el servicio se presta tanto a empresas como usuarios particulares, contando con diferentes tipos de cuentas para ambos tipos de usuario, pero siempre ofreciendo mayores posibilidades a empresas.

Para uso personal, Box ofrece la “Cuenta Personal”, por la que el usuario recibe, gratuitamente, 10GB o, por 8€/mes, 100GB. Esta cuenta permite al usuario opciones básicas y comunes a las demás cuentas más avanzadas, como acceso desde aplicación para smartphone, cifrado SSL y AES, sincronización con aplicación de escritorio.

En el ámbito profesional, Box ofrece más posibilidades.

- Starter: para equipos reducidos, de 1 a 10 integrantes, con un precio de 4€/mes por usuario, con 100GB para cada usuario. Ofrece un historial de 25 versiones para los archivos, así como un control de accesos y estadísticas de uso.

Enterprise: con un precio de 30€/mes por usuario, es la cuenta más personalizable. El almacenamiento es ilimitado igualmente, pero a diferencia de la cuenta business, ofrece acceso y utilización ilimitada a los servicios anteriormente enumerados. También se ofrecen registros de auditorías sobre el contenido de las cuentas.

2.2.2 Desarrollo

Box permite el desarrollo de aplicaciones por parte de terceros. Pone a disposición de desarrolladores dos API's diferentes, con finalidades diferentes. Box Content centra su funcionalidad en el contenido de la cuenta: ficheros, carpetas, metadatos o administración de cuenta y usuarios. Por otro lado, Box View supone una herramienta de conversión de ficheros de Microsoft Office y PDF’s a contenido HTML5 embebible, de forma que se puedan visualizar dichos documentos de forma sencilla, directa y agradable. No existen SDK’s oficiales para esta API.

2.3 Android

Android se define como una plataforma software de código abierto, diseñada e implementada para dispositivos móviles, basado en la versión 2.6 del kernel Linux. Está desarrollado y mantenido por Google, siendo parte de la Open Handset Alliance, una agrupación de empresas de software, hardware y telecomunicaciones, que tiene como objetivo el impulso del desarrollo de estándares abiertos para los dispositivos móviles, siendo Android el pilar sobre el que se erigen todos sus proyectos.

Android es una plataforma de software completa, dado que comprende, además del kernel Linux, una interfaz gráfica de usuario, aplicaciones de usuario, librerías de software, entre otras cosas, explicadas a lo largo de este capítulo. Las aplicaciones se desarrollan en lenguaje Java, y son ejecutadas en una máquina virtual, llamada Dalvik, derivada de la Java Virtual Machine, adaptada y optimizada para entornos móviles.
Actualmente, es la plataforma móvil más extendida del mundo, con cerca del 70% de cuota de mercado en los cinco mayores mercados europeos, a saber, Alemania, Reino Unido, Francia, Italia y España; también domina en otros grandes mercados como China (78,6%), Australia (57,2%).

2.3.1 Historia de Android

En sus inicios, Android era un desarrollo de la compañía Android Inc., fundada por Andy Rubin, hasta que, en 2005, Google la adquirió y continuó con el trabajo sobre Android.

En 2007, nace la Open Handset Alliance, con lo que Android pasa a ser oficialmente open source, dado que el código fuente del proyecto se libera bajo licencia Apache 2.0. Esta licencia posibilita que cualquier persona pueda libremente copiar, modificar o redistribuir dicho código, sin restricciones en la licencia futura.

Durante el año 2008, es liberada la primera versión del SDK de Android, esto es, un conjunto de herramientas, utilidades y librerías que Google facilita a los desarrolladores para que éstos puedan crear fácilmente aplicaciones para el sistema operativo. Poco tiempo después, sale al mercado el primer teléfono móvil con Android, el HTC G1 Dream. Con él, se da a conocer la primera versión estable del sistema operativo, Android 1.0, conocida como Android 1.0 Apple Pie.

En los dos años siguientes, las versiones del sistema operativo se sucedieron y siguieron fabricándose más teléfonos con Android, muchos con una capa de personalización extra realizada por parte de las empresas de hardware. En 2009 Android recibe su primera gran actualización, Android 1.6 Cupcake, con la que llegaban características capitales para el sistema de hoy en día: teclado en pantalla y los widgets, que son mini aplicaciones que proveen determinadas funcionalidades de una aplicación mayor, con la finalidad de facilitar su acceso.

En 2010, sale al mercado el HTC Nexus One que, si bien era manufacturado por HTC, fue diseñado y supervisado estrechamente por Google. El Nexus One no fue un éxito en cuanto a ventas se refiere, pero sí marca un punto de inflexión en la evolución...
del sistema operativo ya que, a partir de entonces, Google impulsa Android aún más, siendo éste el primero de la gama Nexus, cuya característica principal es presentar Android en su versión original, sin personalización extra. También en este año, con la salida de su versión 2.2 Froyo, Android se coloca como 2º sistema operativo móvil más extendido a nivel mundial, sólo por detrás de Symbian.

En 2011, Android da el salto a las tablets, con la versión 3.0 Honeycomb, optimizada para este tipo de dispositivos. A finales de ese año, sale el nuevo Samsung Galaxy Nexus, con la versión Android 4.0 Ice Cream Sandwich, lo que supone una de las mayores evoluciones del sistema a nivel visual.

En 2012 se anuncia que Android alcanza la cifra de 500 millones de dispositivos activados en todo el mundo, consiguiendo, de esta forma, el 75% de la cuota de mercado mundial. Google también decide extender la gama Nexus a las tablets, con la salida de tablet Nexus 7.

Las versiones de Android se siguen sucediendo hasta que, en 2013, se alcanza la cifra de un millón de dispositivos con este sistema operativo.

### 2.3.2 Principales características

A lo largo de sus versiones, Android ha ido incluyendo nuevas características. En esta tabla se muestra un compendio de las más importantes.

<table>
<thead>
<tr>
<th>Diseño de dispositivo</th>
<th>La plataforma es adaptable a pantallas de diferente resolución, VGA, biblioteca de gráficos 2D, biblioteca de gráficos 3D basada en las especificaciones de la OpenGL ES 2.0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Almacenamiento</td>
<td>SQLite, una base de datos liviana.</td>
</tr>
<tr>
<td>Conectividad</td>
<td>GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, WiFi, LTE, HSDPA, HSPA+, NFC y WiMAX, GPRS, UMTS, HSPA+ Y HSDPA+</td>
</tr>
<tr>
<td>Mensajería</td>
<td>SMS y MMS son formas de mensajería, incluyendo mensajería de texto y Google Cloud Messaging for Android (GCM) es parte del servicio de Push Messaging de Android.</td>
</tr>
<tr>
<td>Navegador web</td>
<td>El navegador web incluido en Android está basado en el motor de renderizado de código abierto WebKit, emparejado con el motor JavaScript V8 de Google Chrome.</td>
</tr>
<tr>
<td>Soporte de Java</td>
<td>Aunque la mayoría de las aplicaciones están escritas en Java, no hay una máquina virtual Java en la plataforma. El bytecode Java no es ejecutado, sino que primero se compila en un ejecutable Dalvik y corre en la Máquina Virtual Dalvik. Dalvik es una máquina virtual especializada, diseñada específicamente para Android y optimizada para dispositivos móviles que funcionan con batería y que tienen memoria y procesador limitados. El soporte para J2ME puede ser agregado mediante aplicaciones de terceros como el J2ME MIDP Runner.</td>
</tr>
<tr>
<td>Soporte multimedia</td>
<td>WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), AAC, HE-AAC (en contenedores MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.</td>
</tr>
<tr>
<td>Soporte para hardware adicional</td>
<td>Android soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.</td>
</tr>
<tr>
<td>Entorno de desarrollo</td>
<td>El SDK publicado incluye un emulador de dispositivos, herramientas para depuración de memoria y análisis del rendimiento del software. El entorno de desarrollo integrado es Eclipse usando el plugin de Herramientas de Desarrollo de Android, o bien Android Studio, todavía en estado beta, basado en IntelliJ IDEA.</td>
</tr>
<tr>
<td>Tienda de aplicaciones</td>
<td>Google Play es un catálogo de aplicaciones gratuitas o de pago en el que pueden ser descargadas e instaladas en dispositivos Android sin la necesidad de un PC.</td>
</tr>
<tr>
<td>Multi-táctil</td>
<td>Soporte nativo para pantallas capacitivas con soporte multi-táctil</td>
</tr>
<tr>
<td>Característica</td>
<td>Descripción</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>El soporte para A2DP y AVRCP fue agregado en la versión 1.5; el envío de archivos (OPP) y la exploración del directorio telefónico fueron agregados en la versión 2.0; el marcado por voz junto con el envío de contactos entre teléfonos lo fueron en la versión 2.2. Con la versión 4.3 de Android, se incluye el soporte a Bluetooth 4.0.</td>
</tr>
<tr>
<td>Videollamada</td>
<td>Android soporta videollamada a través de Google Talk desde su versión HoneyComb (3.0).</td>
</tr>
<tr>
<td>Multitarea</td>
<td>Multitarea real de aplicaciones está disponible. Las aplicaciones que no estén ejecutándose en primer plano reciben ciclos de reloj.</td>
</tr>
<tr>
<td>Características basadas en voz</td>
<td>La búsqueda en Google a través de voz está disponible como &quot;Entrada de Búsqueda&quot; desde la versión inicial del sistema.</td>
</tr>
<tr>
<td>Google Now</td>
<td>Google Now es el asistente controlado mediante voz, desde la versión 4.1 Jelly Bean.</td>
</tr>
<tr>
<td>Tethering</td>
<td>Android soporta tethering, que permite al teléfono ser usado como un punto de acces alámbrico o inalámbrico (todos los teléfonos desde la versión 2.2, no oficial en teléfonos con versión 1.6 o inferiores mediante aplicaciones disponibles en Google Play (por ejemplo PdaNet).</td>
</tr>
</tbody>
</table>
2.3.3 Versiones de la plataforma

A lo largo de los años, el sistema operativo ha ido evolucionando desde sus inicios, mejorando características y añadiendo otras. El nombrado de dichas versiones responde un curioso criterio: cada nueva versión llevará el nombre de un postre, en orden alfabético.

![Imagen de versiones de Android]

Figura 3 - Versiones de Android
2.3.4 Arquitectura

Android sigue una arquitectura estratificada en capas. En este tipo de organización, cada uno de los niveles se vale de los inferiores para realizar sus funciones y de esta forma repartir responsabilidades. Es por ello que a este tipo de estructura se conoce como pila. Como se ha resaltado anteriormente, Android se construye sobre un derivado del kernel Linux, en su versión 2.6, sobre el cual se sitúan el resto de capas que facilitan el desarrollo de aplicaciones, puesto que las gestiones de más bajo nivel se dejan en manos de las capas de la arquitectura.

En la figura anterior se puede observar la organización de esta pila, con sus elementos, los cuales se describen en las siguientes secciones.

2.3.4.1 Kernel Linux

De nuevo, todo el conglomerado software que compone Android se dispone sobre una capa base, llamado kernel, que es derivado del kernel Linux 2.6, similar, por tanto, al que puede incluir cualquier distribución de Linux, sólo que adaptado a las necesidades de los dispositivos móviles. Esta capa software proporciona una abstracción de hardware, encargándose de las tareas de bajo nivel a través de los drivers de los
componentes, facilitando la gestión de los componentes físicos. Internamente, las funciones del kernel se dirigen hacia la gestión de la memoria, gestión de procesos, redes y seguridad, entre otras.

2.3.4.2 Librerías

La siguiente capa corresponde con las librerías utilizadas por Android, escritas en el lenguaje C/C++ y proporcionan a Android la mayor parte de sus capacidades y características:

- La librería libc incluye todas las cabeceras y funciones según el estándar del lenguaje C. Todas las demás librerías se definen en este lenguaje.
- La librería Surface Manager es la encargada de componer los diferentes elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.
- OpenGL/ES y SGL representan las librerías gráficas y por tanto sustentan la capacidad gráfica de Android.
- La librería MediaPlayer proporciona todos los códecs necesarios para el contenido multimedia soportado en Android (vídeo, audio, imágenes estáticas y animadas, etc.)
- A través de la librería SQLite, Android ofrece un SGDB ligero, muy adecuado para dispositivos de recursos reducidos, que permite la creación y gestión de bases de datos relacionales.
- La librería WebKit proporciona un motor para las aplicaciones de tipo navegador, y forma el núcleo del actual navegador incluido por defecto en la plataforma Android.

2.3.4.3 Entorno de ejecución

En los sistemas operativos convencionales, las aplicaciones se ejecutan directamente sobre el kernel, pero en Android existe una capa intermedia donde corren las aplicaciones, tanto las integradas en la pila de Android, como las que puedan desarrollar terceros. Esta capa de runtime se compone de las librerías de núcleo, que incluyen gran parte de las posibilidades que ofrece Java; por otro lado, el runtime de
Android integra una máquina virtual similar a la de Java, llamada Dalvik VM, pensada específicamente para entornos móviles.

**Dalvik VM**

Es una evolución de la máquina virtual de Java, diseñada específicamente para Android y los dispositivos móviles, lo que implica unas ciertas restricciones (menor capacidad de procesamiento, memoria reducida, alimentación por batería, etc…). Dalvik delega en el kernel Linux la gestión de los procesos e hilos, la memoria y la gestión de E/S. Por otro lado y a diferencia de Java VM, Dalvik no interpreta directamente el bytecode generado por el compilador Java, sino que trabaja con ficheros .dex que no es más que el bytecode transformado en otro código intermedio más reducido y eficiente. Los ficheros dex pueden combinar una o varias clases del bytecode de Java, generando generalmente menos código.

![Diagrama de Dalvik VM](image)

Figura 5 - Generación código dex y ejecución

Otra de las mayores diferencias con Java VM es que, en lugar de que exista una sola instancia de la VM en ejecución, cada aplicación se ejecute en una instancia independiente de las demás, consiguiendo un impacto muy bajo en la memoria del dispositivo. El principal motivo de este diseño es proteger la ejecución de unas aplicaciones sobre los posibles fallos de las otras.

Para clarificar este punto, sirva de ejemplo un teléfono móvil ejecutando Android, en el que suceda un error que obligue a detener el proceso del navegador web. Con la estrategia tradicional de una única VM en ejecución, este fallo obligaría a la parada de la VM, extendiendo el problema a las demás aplicaciones en ejecución como pueden ser los SMS o el teléfono, lo cual no es aceptable. Gracias a la estrategia de Dalvik, un fallo...
sobre cualquier aplicación permite que las funciones básicas de cualquier teléfono móvil sigan operativas.

Dalvik también gana en eficiencia con respecto a JVM gracias a la utilización de los registros como unidades primarias de almacenamiento, en detrimento de la pila. Por otro lado, Dalvik implementa una recolección de basura más eficiente que JVM.

Por último, cabe destacar que la última actualización del sistema operativo, Android 4.4 KitKat, incluye una nueva máquina virtual, llamada ART, todavía incluida de forma experimental con el propósito de recabar feedback por parte de los desarrolladores y los partners, siendo el objetivo que ART acabe sustituyendo a Dalvik en versiones venideras. La principal diferencia es que ART sustituya la compilación just-in-time (JIT) de Dalvik, que compila el código a dex en tiempo de ejecución y a medida que lo vaya necesitando, por una estrategia ahead-of-time (AOT), que propone una compilación durante la instalación de la aplicación evitando que se compile repetidamente con cada ejecución de la aplicación, transformando ese código en una especie de biblioteca. Consecuentemente, ART aportará un sensible descenso en el uso del procesador, lo que se traduce en una mejora del consumo y de los tiempos de respuesta y ejecución.

2.3.4.4 Framework de aplicación

Esta capa representa, en gran medida, el conjunto de herramientas de desarrollo de cualquier aplicación. Proporciona numerosas librerías Java desarrolladas específicamente para Android, entre ellas, los manejadores de servicios específicos como la localización, sensores o redes WiFi. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo framework. Esta capa está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sueto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean remplazados por el usuario.

Entre las API más importantes están las siguientes:
- **Activity Manager**: Conjunto de API que gestiona el ciclo de vida de las aplicaciones en Android.
- **Window Manager**: Gestiona las ventanas de las aplicaciones y utiliza la librería Surface Manager.
- **Content Provider**: Permite a cualquier aplicación compartir sus datos con las demás aplicaciones de Android. Por ejemplo, gracias a esta API la información de contactos, agenda, mensajes, etc. será accesible para otras aplicaciones.
- **View System**: Proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, mosaicos, botones, "check-boxes", tamaño de ventanas, control de las interfaces mediante teclado, etc. Incluye también algunas vistas estándar para las funcionalidades más frecuentes.
- **Location Manager**: Posibilita a las aplicaciones la obtención de información de localización y posicionamiento.
- **Notification Manager**: Mediante el cual las aplicaciones, usando un mismo formato, comunican al usuario eventos que ocurran durante su ejecución: una llamada entrante, un mensaje recibido, conexión Wi-Fi disponible, ubicación en un punto determinado, etc. Si llevan asociada alguna acción, en Android denominada Intent, (por ejemplo, atender una llamada recibida) ésta se activa mediante un simple clic.
- **XMPP Service**: colección de APIs para utilizar este protocolo de intercambio de mensajes basado en XML.

2.3.4.5 Aplicaciones

La capa más externa de la arquitectura Android es la que componen las aplicaciones de usuario. En este nivel se encuadran tanto las que el usuario instala o compra, como las aportadas de serie en el propio sistema operativo, como pueden ser el teléfono, la agenda de contactos o el navegador. Están escritas en Java, utilizando el SDK que Google proporciona a tal fin. El SDK proporciona herramientas que asisten en el proceso de desarrollo.
Internamente, cada aplicación se ejecuta en un proceso independiente, como se ha descrito en el apartado de la Dalvik VM. Las aplicaciones se empaquetan en ficheros *apk*, que se componen, esencialmente, de archivos dex ejecutables, recursos tales como imágenes o ficheros XML describiendo el diseño de la GUI (layout), y bibliotecas nativas de bajo nivel escritas en C/C++.

Desde el punto de vista de componentes software, una aplicación se compone de cuatro elementos básicos. Cada aplicación será una combinación de uno o más de estos elementos, debiendo ser declarados explícitamente dentro de un archivo xml llamado AndroidManifest.xml, donde también se definen otros datos básicos e indispensables de la aplicación para el sistema operativo, como permisos o valores globales.

**Activity**

Refleja una determinada actividad llevada a cabo por una aplicación. Se asocia típicamente a una ventana o interfaz de usuario, aunque no es una relación obligatoria. Los elementos que se muestran en ella deben ser definidos en el fichero xml que llevan asociado, dentro del cual se definen los elementos tales como ubicación de los elementos en la pantalla (layouts), botones, textos. Las actividades no sólo soportan el aspecto gráfico, sino que éste forma parte del componente Activity. Una aplicación, típicamente, se compone de más de una actividad. El sistema mantiene una pila con las actividades que se van iniciando, de forma que se puedan volver a ellas ordenadamente. Cuando el usuario realiza acciones y provoca cambios de actividad, incluso de aplicación, éstas transitan entre estados del ciclo de vida de la actividad. Los estados que una actividad puede adoptar son los siguientes:

- **Activa** (Running): La actividad está encima de la pila, lo que quiere decir que tiene el foco y es usuario puede interactuar con ella.
- **Visible** (Paused): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla, por ejemplo ocurre en situaciones dónde la aplicación muestra un cuadro de diálogo para interactuar con el usuario. Cuando una actividad está tapada por completo, pasa a estar parada.
- **Parada** (Stopped): Cuando la actividad no es visible. El programador debe guardar el estado de la interfaz de usuario, preferencias, etc.
- **Destruída** (Destroyed): Cuando la actividad termina al invocarse el método `finish()`, o es matada por el sistema.

En cada cambio de estado, se generan eventos en el sistema, de forma que los desarrolladores puedan ejecutar acciones cuando se capturen.

![Ciclo de vida de una actividad](image)
En detalle, los eventos que se generan durante el ciclo de vida:

- **OnCreate**: ocurre cuando se crea la actividad, típicamente donde se construye la interfaz de usuario.
- **OnStart**: muestra la actividad al usuario. Se ejecuta después de OnCreate o OnRestart.
- **OnRestart**: reinicia una actividad tras haber sido parada (si continúa en la pila de tareas). Se inicia desde cero.
- **OnResume**: establece el inicio de la interactividad entre el usuario y la aplicación. Tanto cuando se está creando la actividad por primera vez o cuando se está volviendo a ella.
- **OnPause**: se ejecuta cuando una actividad va a dejar de estar en primer plano, para dar paso a otra.
- **OnStop**: la actividad pasa a un segundo plano por un largo periodo. El sistema puede liberar el espacio que ocupa, en caso de necesidad, o si la actividad lleva parada mucho tiempo.
- **OnDestroy**: paso final del ciclo de vida de una actividad. Se llama cuando ya no es necesaria o cuando voluntariamente se termina mediante llamada del método finish().

**Service**

Los servicios de Android son muy similares en concepto a las actividades. Representan las funcionalidades de las aplicaciones, pero siempre se ejecutan en segundo plano y no disponen de una interfaz gráfica. Por lo general están asociados a una actividad. Un ejemplo claro puede ser el reproductor de música: mientras la lista de canciones, los botones y demás componentes visuales conforman la activity, la reproducción en sí es tarea del service asociado. En el momento que el usuario se mueve de aplicación, la música sigue sonando pero el reproductor deja de ser visible.

El ciclo de vida de los servicios es mucho más simple que el de las actividades y es más controlado por el desarrollador que por el sistema operativo en sí.
Además, el ciclo de vida que sigue un servicio varía en función de la forma en la que quiera ser utilizado:

- **Started services**: pensados para realizar una tarea larga, sin necesidad de interacción con el usuario. Ejemplo: descarga de un fichero de gran tamaño.
- **Bounded services**: diseñados para realizar una tarea en segundo plano, pero con necesidad de interacción con el usuario, estando asociados a una actividad. Ejemplo: reproductor de música, con la reproducción ejecutada mediante el servicio y los controles mediante la interfaz de la actividad.

En función del tipo de servicio, los dos ciclos de vida que puede seguir, son:

![Figura 7 – Ciclo de vida de Service](image-url)
**Content Provider**

Un Content Provider no es más que el mecanismo proporcionado por la plataforma Android para permitir compartir información entre aplicaciones. Por defecto, cada aplicación de Android corre en su propio sandbox, por lo que sus datos son privados para las demás aplicaciones. A pesar de que pequeñas cantidades de información pueden ser intercambiadas entre las aplicaciones mediante intents, los content providers son mucho más adecuados para grandes cantidades de información. Una aplicación que desee que todo o parte de la información que almacena esté disponible de una forma controlada para el resto de aplicaciones del sistema deberá proporcionar un content provider a través del cual se pueda realizar el acceso a dicha información. Este mecanismo es utilizado por muchas de las aplicaciones estándar de un dispositivo Android, como por ejemplo la lista de contactos, la aplicación de SMS, o el calendario/agenda. Esto quiere decir que podríamos acceder a los datos gestionados por estas aplicaciones desde nuestras propias aplicaciones Android haciendo uso de los content providers correspondientes.

Siguiendo el ejemplo de los contactos, cabe señalar que la aplicación de la agenda proporcionada por el sistema Android no posee ninguna lista de contactos propiamente, sino que utiliza este proveedor para utilizar los datos de los contactos. De la misma forma, el proveedor de contactos no tiene una interfaz definida, sólo sirve los datos a aquellas aplicaciones que se lo soliciten. Esta flexibilidad permite que cualquier usuario pueda sustituir la aplicación de agenda del sistema Android por cualquier otra de su elección.

**Broadcast Receiver**

Estos componentes son la implementación de Android del patrón Observer. Un receptor es simple código durmiente que se ejecuta cuando sucede cierto evento al que se ha suscrito. El sistema operativo está constantemente publicando eventos, por ejemplo, la entrada de un SMS o la pérdida de cobertura. Un receptor puede suscribirse a cualquier evento de este tipo y establecer el lanzamiento de una aplicación determinada. Igualmente, una aplicación puede publicar sus propios eventos de forma que sean capturados por los receptores.
Intent

Un intent es un mensaje enviado entre los grandes elementos que conforman las aplicaciones Android, de forma que representan la intención de realizar alguna acción. Pueden ir desde mensajes para abrir una web determinada, abrir una foto, o simplemente cambiar de actividad en la misma aplicación. Un intent es explícito si el creador del mismo especifica cuál es el componente que debe recibir el mensaje, por ejemplo, en el flujo normal de una aplicación que necesite abrir una de sus actividades. Si, por el contrario, únicamente se declara la intención de realizar una acción, sin establecer quién debe hacerla, ese intent será implícito, por ejemplo, abrir una página web. Por lo general, un intent tiene sentido asociado a unos datos específicos, que deben ser adjuntados a éste, de forma que el encargado de ejecutar dicho intent pueda recuperarlos y operar con ellos.

2.4 Android Studio

El entorno de desarrollo elegido para el proyecto será Android Studio. Este IDE está basado en IntelliJ IDEA, por lo que la organización de los proyectos es diferente con respecto a lo propuesto por Eclipse, el otro IDE propuesto por Google para el desarrollo de Android.

2.4.1 Estructura de un proyecto

Un proyecto en Android Studio es una unidad organizativa que representa una solución software completa. En una analogía con Eclipse, se puede decir que el proyecto de Android Studio es un workspace de Eclipse. Un proyecto de Android Studio puede contener varios módulos, que representan unidades concretas de funcionalidad independiente, por lo que pueden ser destinados bien a alojar el código de la aplicación en sí, o bien pueden ser módulos de librería, con código de apoyo. Internamente, un módulo tiene la siguiente estructura de directorios:

- build: directorio de salida de la compilación y empaquetado del proyecto. Contiene las clases compiladas y demás recursos generados durante la compilación.
• **libs**: directorio donde se deben poner librerías externas que necesite el proyecto, por ejemplo, en ficheros jar.

• **src**: contiene tanto como el código de la aplicación, bajo un subdirectorio “main”, como el código de test de la aplicación, bajo “androidTest”. En src/main se organiza el contenido en un directorio “java” para agrupar el código java, y un directorio “res”, donde se almacenarán los ficheros de layout y otros recursos como imágenes.

### 2.4.2 Gradle

Gradle es una herramienta de construcción automatizada de proyectos de toda índole, encargándose de tareas como la construcción, ejecución de pruebas, publicación y despliegue, entre otras cosas. Aúna las posibilidades de construcción de proyectos mediante targets de Ant, con la gestión de dependencias de Maven o Ivy. Su configuración es muy flexible, teniendo disponibles tareas predeterminadas para la mayoría de proyectos contemplados.

Para la programación de tareas en Gradle, se utiliza un “Domain Specific Language” (DSL), basado en Groovy, que es sintácticamente muy similar a Java.

![Figura 8 - Componentes de Gradle](image)

Como muestra de su gran potencia y posibilidades, cabe destacar que proyectos tales como Hibernate, Spring y Grails, utilizan Gradle como sistema de construcción.
3 Análisis Funcional

La sección de análisis funcional describirá los procesos más relevantes que conforman la aplicación Unistor, pretendiendo ser un primer paso en la especificación del sistema desde el punto de vista de más alto nivel. Se darán detalles tanto de la interfaz de usuario (IU), como del proceso en sí.

3.1 Características básicas

3.1.1 Gestión de cuentas

Unistor gestionará una cuenta de Dropbox y una de Box, siendo ambas necesarias, esto es, sólo se podrá iniciar una cuenta de Dropbox y una de Box, nunca dos de uno de los servicios ni sólo una de uno de los servicios. Esto responde a que para una correcta muestra de la gestión común de ambos servicios se considera necesario y suficiente una cuenta de cada servicio. Como consecuencia, en el primer inicio de la aplicación se obligará al usuario a introducir las credenciales de cada uno de los servicios, que se guardarán en las preferencias privadas de la aplicación, de forma que en cada nuevo reinicio de la aplicación, se restauren y no se tengan que volver a pedir. Por otro lado, en caso de que el usuario solicite hacer un logout, se deberá cerrar sesión en ambos servicios, borrando las credenciales guardadas con anterioridad y volviendo a la pantalla inicial de la aplicación.

3.1.2 Operaciones básicas con ficheros

Como operaciones básicas estarán disponibles copiar, mover y eliminar ficheros o carpetas. Estas operaciones serán accesibles a través del menú contextual de la aplicación, mediante pulsación prolongada en el elemento en cuestión. Cabe destacar que no está disponible la selección simultánea de varios elementos, sino de uno sólo.

3.1.3 Subida y descarga de ficheros

El proceso de descarga de ficheros consta de dos pasos diferenciados. Se inicia mediante la comunicación con los servicios de almacenamiento para la obtención del fichero y, una vez obtenida la referencia, se descarga y se almacena en la memoria privada de la aplicación. La segunda fase del proceso es la apertura del fichero con la aplicación predeterminada del sistema para el tipo de fichero. En caso de que no haya
una aplicación determinada y sí varias posibles, se solicitará la elección de una de ellas, aunque esto es más una característica de Android que no de Unistor.

Análogamente, la **subida de ficheros** consta de otros dos pasos. La primera parte cubre la elección de un fichero de la memoria interna del Smartphone del usuario (no la privada de la aplicación), mediante un selector emergente que presentará la estructura de directorios de la memoria y permitirá la navegación en ella y la selección del archivo. La última parte es la comunicación con el servicio de almacenamiento remoto y la consecuente subida del fichero a la nube.

### 3.2 Casos de uso

Un caso de uso es la descripción de una posible secuencia de pasos que llevan a cabo un proceso dentro de la aplicación. Con estos artefactos se acota de forma sencilla y clara las funcionalidades anteriormente detalladas, especificando los actores que toman parte en cada uno de ellos. La siguiente imagen muestra los casos de uso identificados para Unistor.

![Diagrama de casos de uso](image_url)

**Figura 9 - Diagrama de casos de uso**
3.2.1 Especificación de casos de uso

Id: CU001  
Caso de Uso: Iniciar sesión  
Actor: Usuario  
Precondición: -  
Descripción: Solicita los datos de las cuentas de usuario de los servicios e inicia sesión.  

Flujo principal:  
1. Usuario solicita iniciar sesión  
2. Sistema muestra pantalla solicitando los datos de la cuenta de Box.  
3. Usuario introduce datos de cuenta y pulsa aceptar.  
4. Sistema muestra pantalla solicitando los datos de la cuenta de Dropbox.  
5. Usuario introduce datos de cuenta de Dropbox y pulsa aceptar.  
6. Sistema muestra pantalla con los elementos de cada uno de los servicios de almacenamiento.

Id: CU002  
Caso de Uso: Cerrar sesión  
Actor: Usuario  
Precondición: Sesión iniciada  
Descripción: Se cierra la sesión de los servicios.  

Flujo principal:  
1. Usuario solicita cerrar sesión  
2. Sistema cierra sesión y muestra pantalla inicial.

Id: CU003  
Caso de Uso: Navegar entre elementos  
Actor: Usuario  
Precondición: Sesión iniciada  
Descripción: Movimiento entre elementos o carpetas  

Flujo principal: el elemento es una carpeta  
1. Usuario toca sobre una de las carpetas de alguno de los servicios.  
2. Sistema muestra los elementos contenidos en la carpeta y una entrada ofreciendo la vuelta a la carpeta anterior.  

Flujo alternativo: el elemento es un fichero  
1. Usuario toca sobre uno de los ficheros de la carpeta actual.  
2. CU004 Descargar fichero.  

Flujo alternativo: el elemento es “Volver a...”  
1. Usuario toca sobre el elemento “Volver a...”  
2. Sistema muestra los elementos de la carpeta anterior.  
Id: CU004
**Caso de Uso:** Descargar fichero  
**Actor:** Usuario  
**Precondición:** Sesión iniciada y haber tocado un fichero.  
**Descripción:** Se descarga el fichero y se intenta abrir.

**Flujo principal:** Descarga fichero  
1. Sistema muestra diálogo mostrando progreso de descarga.  
2. Sistema abre el fichero con la aplicación más apropiada para cada fichero.

**Flujo alternativo:** Descarga cancelada  
1. Sistema muestra diálogo mostrando progreso de descarga.  
2. Usuario pulsa cancelar en el diálogo.  
3. Sistema oculta el diálogo de progreso e interrumpe la descarga.

**Id:** CU005

**Caso de Uso:** Subir fichero  
**Actor:** Usuario  
**Precondición:** Sesión iniciada.  
**Descripción:** Se sube el fichero seleccionado al servicio.

**Flujo principal**  
1. Usuario solicita subir un fichero.  
2. Sistema abre un diálogo mostrando los elementos del almacenamiento interno del Smartphone.  
3. Usuario selecciona el fichero a subir.  
4. Sistema muestra diálogo con el progreso de la subida.  
5. Sistema muestra oculta el diálogo y muestra mensaje cuando la subida finaliza.

**Id:** CU006

**Caso de Uso:** Copiar fichero  
**Actor:** Usuario  
**Precondición:** Sesión iniciada.  
**Descripción:** Copia de un fichero a otra localización del mismo servicio.

**Flujo principal**  
1. Usuario abre menú contextual.  
2. Usuario solicita copiar un fichero.  
3. Usuario solicita pegar el archivo en el destino.  
4. Sistema muestra el archivo en el nuevo destino.

**Id:** CU007
Caso de Uso: **Mover fichero**  
**Actor:** Usuario  
**Precondición:** Sesión iniciada.  
**Descripción:** Mueve de un fichero a otra localización del mismo servicio.

**Flujo principal**
1. CU006 Copiar fichero  
2. Sistema oculta el fichero en el origen.

**Id:** CU008

Caso de Uso: **Eliminar elemento**  
**Actor:** Usuario  
**Precondición:** Sesión iniciada.  
**Descripción:** Elimina un archivo o carpeta de uno de los servicios.

**Flujo principal**
1. Usuario abre menú contextual.  
2. Usuario solicita eliminar una carpeta o fichero,  
3. Sistema oculta el elemento que acaba de eliminar.
4 Construcción

En esta sección se hablará, de forma introductoria, de la estructura del proyecto que se va a adoptar para esta solución, de forma que se pueda entender más fácilmente los consiguientes diagramas. A raíz de esto, se mostrará el proceso de diseño y desarrollo de la aplicación Unistor. En concreto, se presentarán diagramas de paquetes y de clases para cada uno de esos paquetes. Posteriormente, también se podrán observar diagramas de clases y porciones de código de funcionalidades que satisfacen algunos de los objetivos propuestos. Por último, habrá capturas de la aplicación en ejecución, con las pantallas más importantes para entender el funcionamiento de la misma.

4.1 Estructura del proyecto Unistor

Teniendo presente que el proyecto se llevará a cabo con el IDE Android Studio, la estructura está claramente esbozada. Además, Unistor se articulará como un proyecto multimódulo, con motivo de que la API de Box está implementada como un módulo para Android Studio. La estructura del proyecto, en un primero nivel, es el que sigue:

```
unistor (~/Documents/AndroidStudioProjects/unistor)
├── .idea
├── app
├── BoxAndroidLibraryV2
├── build
├── gradle
│   ├── .gitattributes
│   ├── .gitignore
│   └── build.gradle
├── device-2014-05-20-212401.png
├── gradle.properties
├── gradlew
├── gradlew.bat
├── local.properties
├── references
├── settings.gradle
└── unistor.iml

Figura 10 - Estructura proyecto Unistor
Se pueden apreciar ambos módulos. App es el módulo de código propio de Unistor, la app Android propiamente dicha. Por otro lado, BoxAndroidLibraryV2, es el módulo que Box proporciona para el desarrollo. Es un módulo que funciona como una librería. La carpeta gradle contiene los mecanismos de descarga e instalación de Gradle para el propio proyecto, sin necesidad de que exista en el sistema. La carpeta build, es usada para almacenar las compilaciones de la aplicación. Entre todos los otros ficheros que aparecen, destacan los build.gradle y settings.gradle, que son ficheros de construcción del proyecto de primer nivel, que después se verán particularizados para cada uno de los módulos.

Resulta interesante centrar la descripción en el módulo app, ya que es donde se encuentra el código y recursos del proyecto. La siguiente imagen muestra el contenido de este módulo:
La carpeta principal del módulo es src, dado que abarca todo el código del proyecto. Incluye desde el código Java hasta el código de la interfaz gráfica, definida en XML, pasando por recursos gráficos, como imágenes. Dentro de esta src, la carpeta java contiene las clases Java, organizadas por paquetes, que se explicarán a continuación, en función de su funcionalidad. Por otro lado, la carpeta res contiene los recursos de la aplicación. Las carpetas drawable-XXX contienen las imágenes utilizadas, agrupadas en
función de su densidad (relación de resolución y tamaño). **Layout** contiene los ficheros de la UI de las diferentes pantallas de la aplicación, descritas en XML. El otro directorio importante de la estructura es **values**, que contiene tres archivos:

- **dimens**: un xml para definir dimensiones estándar para reutilizar en los componentes de la aplicación.
- **strings**: xml para definir textos en la aplicación. Este archivo facilita la internalización de la aplicación, simplemente creando un directorio values para cada idioma (Locale), que se definirá en el nombre del fichero, para el caso de francés, por ejemplo, sería values-fr, y dentro un archivo strings con las traducciones.
- **styles**: xml para definir estilos propios en la aplicación.

También, dentro de **src**, se encuentra el AndroidManifest.xml que, como ha sido explicado en el punto dedicado a Android, define datos básicos de la aplicación para el sistema operativo Android, entre ellos, su nombre, activity inicial o permisos necesarios para el funcionamiento de la aplicación.

Para terminal, el archivo build.gradle es el descriptor de compilación del módulo con Gradle. Define ciertos valores, como por ejemplo la versión de Android utilizada o las dependencias que la app tiene.
4.2 Diagrama de paquetes

La organización de los componentes lógicos del proyecto se realiza teniendo en cuenta su funcionalidad y significado. La solución para Unistor consta de cuatro paquetes:

- **activity**: alberga aquellas clases que implementan las pantallas de la aplicación u otros componentes gráficos de mayor entidad. Las clases que se encuentran en esta paquete extienden la clase Activity (o alguna de sus subclases), o bien extienden la clase Fragment, que componen diferentes vistas dentro de la misma Activity.

- **adapter**: contiene las clases que implementan las listas de elementos de las pantallas de la aplicación, esto es, son subclases de Adapter, así como aquellas clases de apoyo, como las que representan cada uno de los elementos de la lista. En este proyecto, el Adapter implementado es la lista de elementos que conforman la cuenta del usuario en cada uno de los servicios.

- **asynctask**: el funcionamiento y comunicación de la aplicación con los servicios web de Dropbox y Box se ha de hacer en segundo plano, dado que las llamadas
podrían no ser inmediatas. Es por esto que la aplicación se valdrá de las clases que Android dispone a tal fin, implementando la interfaz AsyncTask.

- **util**: clases de apoyo necesarias para el funcionamiento de la aplicación, que no encajaban en ninguno de los otros bloques lógicos. Van desde efectos gráficos a clases POJO.

El diagrama no sólo muestra los paquetes que orquestan la aplicación, sino también las relaciones entre ellos. En este caso, las relaciones entre paquetes se establecen como dependencias simples, dado que el tipo de relación entre clases se especificará en el apartado dedicado a cada paquete.
4.3 Diagrama de clases

4.3.1 Paquete activity

Retomando lo anteriormente mencionado, los componentes Activity son los principales componentes visuales de las aplicaciones Android. En Unistor, solamente existirá una Activity como tal, MainActivity, que aglutinará la vista para las cuentas abiertas. Se puede entender como un contenedor que, bien puede contener un panel simple con controles, imágenes y demás, o bien un conjunto de subvistas o Fragment, donde unas pueden ser visibles y otras ocultas. La segunda aproximación es la relevante en Unistor.

Los tipos de cuentas tendrán forma de jerarquía de clases Fragment, encabezada por UnistorFragment, dado que las operaciones relacionadas con el comportamiento visual, como la presentación y el refresco de la lista de elementos de la cuenta, serán comunes para cualquier tipo de cuenta que forme parte de este proyecto o de los que se puedan incluir en el futuro. Por otro lado, hay ciertas operaciones que dependen de las peculiaridades de cada servicio de almacenamiento y de su API, por lo que se
implementan en cada clase hija, **DropboxFragment** y **BoxFragment**. Esta solución aporta sencillez a la hora de manejar operaciones comunes y flexibilidad para aquellas que necesiten un grado de especialización mayor.

Cabe destacar que MainActivity no gestiona directamente los Fragment, sino que se apoya en otros elementos, en forma de composición, para llevarlo a cabo. Se compone de la clase **ViewPager**, que se encarga de la gestión de los Fragments utilizados como vistas, además de permitir que la presentación de los mismos sea a través de gestos de deslizar páginas; y **SectionPagerAdapter**, encargado de crear cada una de las vistas a partir de los datos que se le faciliten. Es esta clase, SectionPagerAdapter, la que se encarga de mantener los Fragment de la jerarquía y decidir cuál mostrar en cada momento.

![Figura 14 - Representación de Swipe entre Fragments](image)

Son todos estos elementos los responsables de recibir las acciones del usuario y ejecutar las funcionalidades asociadas a esas acciones, interactuando entre ellas, típicamente entre MainActivity y los Fragments, y con otras clases de los demás paquetes.
Por último, se ha definido una interfaz, OnUploadFinishListener, que haga las veces de callback que ejecuta un determinado código cuando la subida del fichero acaba.

### 4.3.2 Paquete adapter

Cada uno de los Fragment del paquete activity contendrá una lista donde se presentarán los elementos que conforman las cuentas de usuario de los servicios Dropbox y Box, a través de un elemento visual, ListView, proporcionado en el sdk de Android. Listview muestra los elementos en una lista vertical deslizable. Por otro lado, los datos de la lista se mantienen en un Adapter, que es el intermediario. Para este proyecto se ha creado **UnistorEntryListAdapter**, que almacena los datos en forma de conjunto de objetos **UnistorEntry** que, a su vez, encapsula los datos de cada una de las entradas. Esta clase tiene una particularidad, dado que implementa la interfaz Parcelable. Esto facilita que este tipo de clases sean intercambiadas entre, por ejemplo, actividades y que se puedan almacenar en, por ejemplo, preferencias de la aplicación.

Este adaptador no sólo se encarga de servir los datos al elemento visual, sino que también tiene la responsabilidad de crear la vista para cada uno de los ítems en la lista, esto es, crear cada uno de los elementos que conforman la vista (TextView, imágenes, checkbox, etc...) y darles el valor correspondiente en cada caso. Esto se realiza en el método getView de la clase UnistorEntryListView. Aunque la operativa de este método...
se presentará en las siguientes secciones, es notable que se utiliza un patrón de diseño llamado ViewHolder. Este patrón especifica que se debe crear una clase, ViewHolder en este caso, que almacena cada componente de la vista (entrada de la lista), de forma que se puedan modificar directamente sus valores sin la necesidad de tener que volver a buscar la referencia. Esto resulta en una mejora sustancial en el scroll de la lista, puesto que los tiempos de creación de cada elemento de la lista se minimizan.

### 4.3.3 Paquete asynctask

![Diagrama clases paquete AsyncTask](image)

Según lo planteado anteriormente, este paquete aglutina las clases que realizan trabajos de comunicación con los servicios de Dropbox y Box y que podrían implicar un tiempo elevado, con lo que estas operaciones deben desligarse del hilo principal de la UI. Para esta operación, el SDK de Android proporciona la clase AsyncTask, que facilita este desacoplamiento. De esta forma, mientras se lleva a cabo la comunicación con los servidores, no se bloqueará la aplicación, permitiendo al usuario seguir interactuando con la UI.

Las operaciones que más tiempo pueden consumir en la aplicación son la descarga y la subida de ficheros. Las clases DownloadFileAsyncTask y UploadFileAsyncTask, respectivamente, se han creado para implementar dichos procesos, tanto para ficheros que procedan de Dropbox como para ficheros alojados en Box. Se ha optado por diseñar las clases gestionando ambos servicios debido a que las diferencias de procedimiento
en función del servicio de almacenamiento son mínimas, sólo varía la comunicación mediante uno u otro API, siendo invariable el resto del procedimiento, a saber, el guardado del fichero descargado en almacenamiento local o la selección del fichero a subir.

Estas clases asíncronas se encargan del grueso del trabajo, pero se apoyan en otras clases para gestionar el avance del proceso y, de esta forma, dar feedback al usuario. BoxProgressListener se encarga del avance en caso de un proceso de Box y, análogamente, DropboxProgressListener se encarga del avance de un proceso de Dropbox. Ambas implementan sendas interfaces proporcionadas por los SDK’s de los servicios de almacenamiento, IFileTransferListener para Box y ProgressListener para Dropbox.

Las relaciones de las tareas asíncronas con estas clases son de composición debido a la fuerte dependencia entre ellas. Es obvio que ambos progressListener sólo tienen sentido mientras una transferencia (subida o bajada) está ocurriendo, de la misma forma que también es indispensable que una transferencia tenga un progressListener asociado para poder comunicar los progresos. Todo esto pone de manifiesto que ambos deben permanecer fuertemente ligados, lo que se corresponde con una composición.

4.3.4 Paquete útil

Figura 17 - Diagrama clases paquete Util
El último de los paquetes, será el paquete `util`, que contiene clases de apoyo para ciertos procesos que el proyecto ejecuta para cumplir los objetivos y que no encajan funcionalmente en otros paquetes.

El primer elemento a destacar podría ser el **TmpContentProvider**, que es un elemento que implementa la interfaz Content Provider, uno de los componentes software que conforman Android. Los content provider sirven información entre aplicaciones, salvando el aislamiento en el que Android ejecuta cada aplicación. Es necesario este artefacto en Unistor debido a que los ficheros descargados se almacenan en el espacio privado de cada aplicación, por lo que a la hora de abrir un archivo, es necesario algún mecanismo que “publique” este fichero, de forma que las aplicaciones puedan ver y abrirlo en última instancia. TmpContentProvider cubre esta necesidad en Unistor.

La clase **ContentStatus** representa el contenido de la carpeta que en cada momento se esté mostrando, es decir, el conjunto de elementos en la lista de los Fragment. Internamente, no es más que una lista de **UnistorEntry**, del paquete adapter. La razón de su existencia, es facilitar la vuelta atrás en la navegación entre carpetas. De esta forma, se reduce la llamada a servicios y, en consecuencia, el consumo de datos. También cabe destacar que esta clase implementa la interfaz Parcelable, ya explicada en el paquete adapter. La razón es que también será usada para almacenar el estado de la aplicación en el momento de un cierre de la misma y, de esta forma, poder restaurarlo en la siguiente apertura.

También está en este paquete la clase **UnisterEntryComparator**. Esta clase, que implementa la interfaz Comparator, tiene como finalidad comparar los elementos contenidos en una ruta concreta, de forma que se puedan ordenar en la lista de una manera lógica. En este orden de elementos, las carpetas siempre irán primero, y después el resto de ficheros, independientemente de su extensión. Entre elementos del mismo tipo, prevalecerá un orden alfabético.

**SimpleFileDialog** aporta la funcionalidad de la elección del fichero para la subida a los servicios de almacenamiento. Es un elemento integrado en Unistor, pero obtenido de un proyecto de código abierto de Scorch Works. SimpleFileDialog busca la ruta exacta
del almacenamiento externo del Smartphone y construye una lista de elementos contenidos en él. Permite la navegación entre carpetas y la selección de un único fichero simultáneamente. Una vez seleccionado el fichero, se podrá especificar las acciones a realizar con él, por lo que la reutilización de este código está garantizada.

Una de las clases que más se alejan de la responsabilidad funcional de la aplicación es ZoomOutPageTransformer, que se encarga de aportar un efecto óptico en la transición entre los UnistorFragment. El código de esta clase ha sido obtenido de la web de recursos del programador de Android.

Por último, la clase Constants aglutina gran parte de las constantes utilizadas en la aplicación. Facilita la reutilización y el control de estos valores constantes en diversos lugares de la aplicación.

4.4 Implementación

4.4.1 Descarga de ficheros

Siguiendo lo especificado en el diagrama de clases del paquete asynctask, la clase DownloadFileAsyncTask se encargará de la descarga en segundo plano de ficheros de las cuentas de almacenamiento online, valiéndose de la clase AsyncTask facilitada por el SDK de Android. El esqueleto de la misma es el que sigue:

```java
public class DownloadFileAsyncTask extends AsyncTask<Void, Long, Boolean> {
    private final String TAG = "DownloadFileAsyncTask";
    private ProgressDialog mProgressDialog;
    private Context mContext;
    private String mFileName;
    private String mErrorMsg;
    /* Dropbox parameters */
    private DropboxAPI<? mDBApi;
    private String mRemoteFilePath;
    private FileOutputStream mOutputStream;
    /* Box parameters */
    private BoxAndroidClient mBoxClient;
    private String mFileID;
    private double mFileSize;
    private BoxProgressListener mBoxProgressListener;

    @Override
    protected void onPreExecute() {
    }
```
En la declaración de la clase se ve especificada la relación de herencia que mantiene con la clase AsyncTask. De ella hereda los métodos que, posteriormente, deberán ejecutar el código responsable de la funcionalidad de la descarga de ficheros.

Cabe destacar que AsyncTask es una clase genérica con tres parámetros. El primero es el argumento que se le deberá enviar a la clase en el momento de su ejecución y que, posteriormente, recibirá el método doInBackground. En el caso de DownloadAsynctask, no se utilizará esta forma de enviar parámetros, ya que se necesitan más y de diferentes tipos, por lo que se utilizarán los constructores de clase para enviarlos. El segundo parámetro indica el tipo de dato que describirá el progreso de la tarea. En este caso, se describirá mediante una escala de 0 a 100. El tercer y último argumento es el resultado de la ejecución doInBackgound que, a su vez, será la entrada que recibirá onPostExecute.

En la primera sección de la clase se definen los atributos internos de la misma, diferenciando los atributos necesarios para operar con cada uno de los servicios y los atributos comunes, necesarios para ambos. Entre los atributos se encuentran los objetos que encapsulan las APIs de los servicios de Dropbox y Box, DropboxAPI y BoxAndroidClient, respectivamente. Estos objetos serán utilizados a la hora de ejecutar la comunicación y solicitar la descarga del fichero. También se define un atributo de tipo
ProgressDialog, que será utilizado para mostrar un cuadro de diálogo donde se mostrará el progreso de la tarea (la descarga), dando opción de cancelar el proceso.

La segunda parte de la clase contiene los métodos de la clase, tanto los heredados, que habrá que sobrescribir, como otros creados específicamente. Toda clase asíncrona en Android tendrá el mismo ciclo de vida, es decir, su flujo de ejecución está definido. La siguiente imagen resume su funcionamiento.

Al crearse el objeto de la clase AsyncTask se invoca, para empezar, a su primer método, onPreExecute, que se ejecuta sobre el hilo principal. Al terminar éste, se crea un hilo secundario y ejecuta su trabajo dentro de doInBackground. Durante la ejecución en segundo plano podemos realizar llamadas al hilo principal (por ejemplo, para incrementar la barra de carga del ProgressDialog a medida que se ejecuta el hilo en segundo plano) desde doInBackground, y con ayuda de publishProgress, a un método sobrescrito llamado onProgressUpdate. No es obligatorio llamar a publishProgress, pudiendo no llamarlo nunca con lo que nunca entraremos en onProgressUpdate, o llamándolo las veces que queramos. Al terminar de ejecutarse doInBackground se llama de inmediato a onPostExecute, donde se tratará el resultado del trabajo, dándose por acabado a este hilo en segundo plano.

En la clase asíncrona DownloadAsyncTask, todos los métodos arriba comentados tendrán una función definida. Siguiendo el flujo, el código de onPreExecute:

```java
@Override
protected void onPreExecute() {
    super.onPreExecute();

    // Construcción del ProgressDialog, con el que se informará
    // al usuario del avance de la tarea.
    String btnLbl = mContext.getString(R.string.btn_cancel);
    mProgressDialog = new ProgressDialog(mContext);
    mProgressDialog.setMessage(mContext.getString(R.string.download_dialog_msg) + mFileName);

    mProgressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
```
En este paso previo a la ejecución del trabajo, se crea el ProgressDialog que dará la información del progreso de la tarea. Se le da un título, se define el estilo visual (una barra horizontal de progreso) y se establece que es cancelable (setCancelable) mediante el uso del botón “Atrás” del sistema operativo. También se habilita un botón “Cancelar” en el diálogo y se establecen las acciones a realizar cuando, mediante alguna de las dos posibilidades, se cancela el diálogo: se cancela la tarea asincrónica y se muestra un mensaje informando de la cancelación.

El siguiente paso, es la ejecución del trabajo, en el método doInBackground.
En este método se realizará la descarga del archivo, su almacenamiento en la memoria privada de la aplicación y se solicitará la apertura del mismo mediante el Content Provider implementado. Dado que son dos los servicios de almacenamiento que se deben gestionar, la primera acción será determinar el servicio al que dirigir las peticiones (el objeto API será no nulo). Posteriormente, se realizará la descarga, a través de los métodos privados `downloadBoxFile` y `downloadDropBoxFile`, que contendrán la comunicación con cada uno de los servicios.

```java
private File downloadBoxFile() {
   // Comprobación del estado de la tarea
   if (isCancelled()) {
      return null;
   }

   // Creación del fichero local, inicialización del listener que gestorará el progreso
   // y comunicación con el servicio a través del API
   File f = new File(mContext.getFilesDir(), this.mFileName);
   this.mBoxProgressListener = new BoxProgressListener();
   
   return f;
}
```
try {
    mBoxClient.getFilesManager().downloadFile(this.mFileID, f, this.mBoxProgressListener, null);
    //TODO error handling
    } catch (Exception e) {} 

    return f;
}

private DropboxAPI.DropboxFileInfo downloadDropBoxFile(){
    DropboxAPI.DropboxFileInfo file = null;
    try {
        // Comprobación del estado de la tarea
        if(isCancelled()){ 
            return null;
        }

        // Se crea un flujo de salida para almacenar el fichero local.
        mOutputStream = mContext.openFileOutput(this.mFileName, Context.MODE_PRIVATE);

        // El fichero es descargado a través del API de Dropbox.
        // MyProgressListener maneja el progreso
        file = mDBApi.getFile(mRemoteFilePath, null, mOutputStream, new DropboxProgressListener());

        Log.i("Info: ","Download Completed!");
    } catch (...){
    } finally {
        try {
            mOutputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    return file;
}

Dado que ambos tienen la misma finalidad, la estructura es similar. La diferencia más destacable es que, en caso de Dropbox, se opta por descargar el archivo en forma de OutputStream, dado que se ofrece la posibilidad de un mayor control del progreso del proceso, mediante una instancia de la clase DropboxProgressListener, para después almacenarlo en un fichero interno. En caso de Box, se almacena directamente en un fichero local, pues el API de Box realiza este paso intermedio de forma transparente a la vez que ofrece, también, la posibilidad de monitorizar el progreso con una instancia de la clase BoxProgressListener.

Como inciso, cabe destacar que, en última instancia, las API’s consumen servicios web de tipo RESTful, por lo que, aunque enmascaren estas llamadas, se deben tratar los
posibles resultados de error los servicios, en forma de excepciones. Para ahorrar en espacio y ganar en claridad, este código no se ha incluido en la memoria.

Para terminar el ciclo de la tarea asincrónica, se ejecuta onPostExecute que termina el dialogo de progreso y, en función del resultado de doInBackground, mostrará un mensaje de éxito o no.

```java
@Override
protected void onPostExecute(Boolean result) {
    mProgressDialog.dismiss();
    if(result){
        Toast.makeText(mContext,R.string.download_completed,Toast.LENGTH_LONG).show();
    }
}
```

4.4.2 Presentación de elementos de cuenta

Esta parte de la implementación pretende mostrar los elementos contenidos en una carpeta, una ruta particular de las cuentas, en forma de lista desplazable, de forma que la navegación sea sencilla e intuitiva. Android aporta un componente visual que esboza estas capacidades, ListView. Este elemento se vale de un Adapter para obtener los datos a presentar en la UI, acomodándolos en el layout que se especifique. Existe la posibilidad de mostrar los elementos en listas simples, de sólo texto, pero también la posibilidad de definir un layout personalizado y construir la interfaz ítem a ítem, decidiendo qué contenido irá en cada elemento de la misma, pudiendo ir desde simple texto a selectores, pasando por imágenes. En el caso de Unistor, se ha definido la siguiente interfaz para cada una de las entradas del ListView:

```xml
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingLeft="15dp"
    android:paddingRight="15dp"
    android:descendantFocusability="beforeDescendants">

    <RelativeLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingTop="10dp"
        android:paddingBottom="10dp">
```
android:background="@drawable/card_background_selector"
android:descendantFocusability="afterDescendants">

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="@android:attr/textAppearanceMedium"
android:id="@+id/entryName"
android:textColor="#ff0b3b76"
android:paddingRight="2dp"
android:ellipsize="middle"
android:singleLine="true"
android:text="Esto es una entrada"
android:layout_above="@+id/entrySize"
android:layout_toRightOf="@+id/entryIcon" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="@android:attr/textAppearanceSmall"
android:id="@+id/entrySize"
android:textColor="#87000000"
android:paddingRight="2dp"
android:ellipsize="middle"
android:singleLine="true"
android:text="5 MB"
android:layout_marginRight="12dp"
android:layout_alignBottom="@+id/entryIcon"
android:layout_alignParentRight="true"
android:layout_alignParentEnd="true"
android:textStyle="italic" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="@android:attr/textAppearanceSmall"
android:text="Small Text"
android:id="@+id/lastModification"
android:layout_marginRight="5dp"
android:layout_alignTop="@+id/entrySize"
android:layout_toLeftOf="@+id/entrySize"
android:textStyle="italic"
android:textColor="#87000000" />

<ImageView
android:layout_width="48dp"
android:layout_height="48dp"
android:id="@+id/entryIcon"
android:layout_centerVertical="true"
android:layout_margin="8dp"
android:contentDescription="@string/desc_icon"
android:src="@drawable/ic_launcher"/>

</RelativeLayout>
</FrameLayout>
La estructura de la entrada consta de:

- Nombre de la entrada: TextView.
- Tamaño del elemento: TextView.
- Última moficación: TextView.
- Icono: ImageView.

Todos estos elementos se disponen dentro de un RelativeLayout, lo que indica que el posicionamiento de los elementos se realiza en relación con sus hermanos y padre. Los atributos clave para esto son:

<table>
<thead>
<tr>
<th>Atributo</th>
<th>Significado</th>
</tr>
</thead>
<tbody>
<tr>
<td>android:layout_above</td>
<td>Sitúa el elemento sobre elemento de id especificado.</td>
</tr>
<tr>
<td>android:layout_toRightOf</td>
<td>Sitúa el elemento a la derecha de elemento de id especificado.</td>
</tr>
<tr>
<td>android:layout_toLeftOf</td>
<td>Sitúa el elemento a la izquierda del elemento de id especificado.</td>
</tr>
<tr>
<td>android:layout_alignBottom</td>
<td>Sitúa el elemento debajo de elemento de id especificado.</td>
</tr>
<tr>
<td>android:layout_alignParentRight</td>
<td>Alinea en el lado derecho del padre.</td>
</tr>
<tr>
<td>android:layout_alignParentEnd</td>
<td>Alinea en el vértice inferior derecho del padre.</td>
</tr>
<tr>
<td>android:layout_centerVertical</td>
<td>Alinea en el centro del padre.</td>
</tr>
</tbody>
</table>

En este punto, se encuentra definida la UI para los elementos de la lista de elementos de cada servicio. El siguiente paso para dar forma a esta lista sería construir cada una de las entradas. Para esto se ha construido un Adapter personalizado. Un adapter en Android se puede decir, es una colección de datos, que se asignan a una vista para que ésta los muestre. Creando un adaptador personalizado existe la posibilidad de crear la vista exactamente como se desee. Para ello, se ha definido una clase, UnistorEntryListAdapter, que contendrá la lista de elementos UnistorEntry y que construirá la vista de cada fila, en el método heredado getView.
super(context, R.layout.dropbox_list_item, itemList);

mContext = context;

}/*
 * El argumento convertView representa la fila a
 * crear/modificar
 * en cada momento. Tendrá un valor no nulo
 * si se trata de una fila ya creada y que se vaya a
 * reutilizar.
 * Por tanto, cuando un convertView es no nulo, simplemente
 * se actualiza el contenido de su layout.
 */
@Override
public View getView(int position, View convertView, ViewGroup parent) {

/*
 * Mediante el uso del patron ViewHolder, el número de llamadas a
 * findViewById se reduce manteniendo referencias a todas las
 * vistas de
 * una fila. Por esto, findViewById es sólo invocado la primera vez
 * que
 * se crea la fila.
 */

ViewHolder holder;

if(convertView==null)

    convertView = LayoutInflater.from(mContext)
          .inflate(R.layout.dropbox_list_item, null);

    holder = new ViewHolder();
    holder.setIcon(((ImageView)convertView
          .findViewById(R.id.entryIcon)));
    holder.setName(((TextView)convertView
          .findViewById(R.id.entryName)));
    holder.setSize(((TextView)convertView
          .findViewById(R.id.entrySize)));
    holder.setLastModification(((TextView)convertView
          .findViewById(R.id.lastModification)));
    convertView.setTag(holder);

else {

    holder = (ViewHolder)convertView.getTag();
}

Log.i("UnistorEntryListAdapter.getView",
      this.getItem(position).getName());

holder.getName().setText(this.getItem(position).getName());

holder.getIcon().setImageResource(this.getItem(position)
      .getEntryIcon(mContext));

holder.getSize().setText(this.getItem(position).getSizeString());

holder.getLastModification()
      .setText(this.getItem(position).getLastModification());
holder.setEntry(this.getItem(position));

return convertView;
}
}

En el constructor de la clase, se define que el layout a utilizar es, precisamente, el creado anteriormente para la representación de los elementos de la lista.

En el método getView, como se ha dicho, se crea la vista de cada una de las filas. Para ello, el método tiene un argumento, convertView, que representa la vista de la fila. Este es, por tanto, el elemento a modificar para la construcción de la lista de elementos. El funcionamiento básico del método consiste en coger las referencias de los elementos visuales que conforman la fila y darles un contenido con el elemento de la posición del parámetro position. Normalmente, este método realiza operaciones muy costosas, como la creación del layout, layoutInflater.from(mContext).inflate(R.layout.dropbox_list_item, null). Esto puede provocar un incómodo lag en cuanto se intente desplazar la lista, debido a que el método se ejecutaría cada vez que se actualizan las filas visibles y ocultas. Para esto, se ha utilizado un patrón de diseño, llamado ViewHolder, que propone la creación de un objeto que mantenga permanentes referencias a los elementos de las vistas de cada fila, de forma que la creación sólo deba realizarse la primera vez y, en las posteriores actualizaciones, únicamente se deba actualizar el contenido de dichas vistas. Al final, un ViewHolder no es más que un JavaBean con las vistas de la fila como propiedades:

```
package cen.unistor.app.adapter;

public class ViewHolder{
    private TextView name;
    private ImageView icon;
    private TextView size;
    private TextView lastModification;
    private UnistorEntry entry;

    public TextView getName() {
        return name;
    }

    public void setName(TextView name) {
        this.name = name;
    }

    public ImageView getIcon() {
        return icon;
    }
}```
public void setIcon(ImageView icon) {
    this.icon = icon;
}

public UnistorEntry getEntry() {
    return entry;
}

public void setEntry(UnistorEntry entry) {
    this.entry = entry;
}

public TextView getSize() {
    return size;
}

public void setSize(TextView size) {
    this.size = size;
}

public TextView getLastModification() {
    return lastModification;
}

public void setLastModification(TextView lastModification) {
    this.lastModification = lastModification;
}

Este objeto se deberá almacenar con la vista convertView, que será la diferente para cada fila, por tanto nunca se sobrescribirá. Esto permite que, la siguiente iteración sobre la misma fila, pueda ser recuperado fácilmente y de forma muy ligera.
5 Conclusiones

A lo largo del presente documento, se ha presentado la aplicación Unistor, una aplicación Android de gestión unificada de servicios de almacenamiento online. No es una solución novedosa, puesto que existen otras aplicaciones más completas destinadas a cada uno de los servicios, pero no tantas que gestionen varios de forma conjunta. Se han introducido los servicios a utilizar, así como el sistema operativo sobre el que se ejecutará la aplicación. Se ha realizado un proyecto muy concreto con funcionalidades claras y sencillas, enfocadas a la demostración de las posibilidades de una interfaz unificada para diferentes servicios de almacenamiento online, sin pretender ser una solución completa.

Las pretensiones eran mostrar el uso de diferentes servicios de almacenamiento en una misma aplicación, habiendo conseguido este objetivo al ofrecer todas las funcionalidades planteadas en la introducción.

En esta primera aproximación se han utilizado sólo dos servicios. En estos dos puntos se concentran las líneas futuras de ampliación de este proyecto:

- Añadir nuevos servicios como Google Drive o Microsoft OneDrive.
- Gestión de cuentas mejorada: individualizar el inicio de sesión y posibilidad de gestionar varias cuentas del mismo servicio.
- Crear carpetas.
- Renombrar ficheros y carpetas.
- Copiar o mover ficheros y carpetas entre distintos servicios.
Bibliografía


