Universidad Politécnica de Valencia
Escuela Técnica Superior de Ingeniería Informática
Ingeniería Técnica en Informática de Sistemas

Desarrollo de una Aplicación Móvil para la Gestión de Tareas Personales

ALUMNO **Sergio Segarra Miró**
DIRECTOR **Vicente Pelechano Ferragud**
CODIRECTORES **Pablo Muñoz Julve y Miriam Gil Pascual**

Septiembre 2011

A Cris.

A mis padres y a mi hermano.

Al Centro PROS por su confianza y apoyo, en especial a Pablo Muñoz.

Resumen

Los dispositivos móviles se están convirtiendo en la principal herramienta de comunicación en la vida cotidiana de las personas. Su evolución ha permitido la implantación de potentes sistemas operativos y la proliferación de aplicaciones y servicios digitales que pueden utilizarse desde cualquier lugar. A su vez, los desarrolladores están creando aplicaciones que cubren parte de esas nuevas funcionalidades que inicialmente no se incluyen en los dispositivos.

Entre la enorme variedad de aplicaciones y sus distintas funcionalidades encontramos las aplicaciones para gestionar tareas. En este ámbito se observa la falta de integración de este tipo de aplicaciones con el contexto del usuario y las redes sociales.

En este trabajo se presenta una propuesta para cubrir esta carencia mediante la incorporación del contexto del usuario en la gestión de los avisos y alarmas de la aplicación, con el objetivo de no molestar al usuario. De esta forma, se realiza una distinción entre las alarmas de las tareas según su prioridad, con el objetivo de que las alarmas más prioritarias tengan avisos que demanden mayor grado de atención del usuario, y las menos prioritarias, avisen de forma más sutil demandando menos atención.

Se implementa además, la integración de la aplicación con la red social Facebook. Una vez iniciada la sesión, es posible descargar los eventos propios del usuario para que formen parte del listado de tareas.

Finalmente, se presenta una aplicación completa de gestión de tareas para dispositivos móviles diseñada e implementada bajo la plataforma Android.

Índice de Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1. Introducción

En la actualidad, un sector de dispositivos móviles ha evolucionado para dejar en un segundo plano primitivas funcionalidades como la realización de llamadas o el envío de mensajes de texto o multimedia. Estos dispositivos más avanzados, acuñados con el término inglés Smartphone, se acercan cada vez más al concepto de ordenador personal, con unos componentes hardware más potentes que hacen posible la ejecución fluida de un sistema operativo específico.

En los últimos meses, Apple y Google liberan una batalla software con sus sistemas operativos específicos para dispositivos móviles. En el caso de Apple, iOS está diseñado específicamente para dispositivos Apple, con un hardware específico y concreto como son iPhone, iPod o iPad. Su efectividad y fluidez es debida en parte a que la programación de sus aplicaciones se realiza en Objective-C, una evolución del lenguaje C, que trabaja sobre el núcleo del sistema [2]. El entorno de desarrollo de Apple requiere un Mac para su instalación, además de estar registrado como desarrollador de esta compañía. Por último, para su venta y distribución es necesario pagar una licencia de 99$ [3]. Por otro lado, Android es el sistema operativo de Google para dispositivos móviles, aunque su uso se ha extendido a reproductores de MP3, portátiles ligeros, o incluso en ordenadores de sobremesa [10]. Ha sido diseñado para distintos dispositivos de otros fabricantes y por lo tanto, preparado para una gama de componentes hardware más amplia que iOS. Fabricantes como HTC, Motorola o Samsung han adoptado a Android como principal sistema operativo de su catálogo de dispositivos móviles.

Estamos asistiendo, por tanto, a una revolución de Smartphones que están cambiando nuestros hábitos y formas de trabajo y comunicación. Así, el software de un Smartphone (tanto su sistema operativo como sus servicios que éste puede ofrecer) se ha convertido en una parte muy importante en estos dispositivos.

Los Smartphone se han integrado en el entorno social como una herramienta indispensable de comunicación. Con esta conectividad permanente a internet, el usuario puede por ejemplo participar en redes sociales, crear grupos de amistad, o utilizar aplicaciones de mensajería instantánea y de correo electrónico. Los dispositivos proveen al usuario de aplicaciones predeterminadas tales como el correo electrónico, navegador web, redes sociales, reproductores de música o incluso navegadores por GPS.

Además de este tipo de servicios, los desarrolladores están creando aplicaciones que cubren una gran variedad de funcionalidades añadidas que inicialmente no incluyen los dispositivos. Una de éstas son las aplicaciones que permiten gestionar tareas y llevar un control de los trabajos a realizar en un tiempo

limitado. En este proyecto se desarrolla una aplicación de este tipo, personalizable a los recursos de atención del usuario.

1.1. Aplicaciones de Listas de Tareas

Una lista de tareas es un conjunto de cosas pendientes de hacer. Deben ser brevemente descritas de tal forma que al leerlas, un usuario sepa toda aquella información necesaria para completar estas tareas [1].

Un ejemplo sencillo de lista de tareas que es posible encontrar en la vida real es el de una libreta pequeña con anotaciones de tareas que se deben realizar en un periodo concreto de tiempo como pueden ser por ejemplo ir al supermercado o devolver un libro en una biblioteca. Cuando una tarea se ha realizado es tachada de la lista al ser finalizada.

En el ámbito digital una aplicación que permita gestionar una lista de tareas funciona del mismo modo y cuenta con las siguientes características básicas. Es posible que pueda contar con varias listas, o categorías que organizan por temáticas las tareas. Además, cada tarea dispone de un título corto que la identifica, una fecha y hora de vencimiento antes de la cual debe terminarse y una alarma asociada que avise al usuario que finaliza el plazo para su realización. Las acciones principales de una lista de tareas son la inserción, edición, y finalización de tareas. Además, para ocultar las tareas ya finalizadas se utiliza la eliminación de tareas del gestor.

1.2. Nuestro Proyecto de Listas de Tareas

Para este proyecto se ha diseñado e implementado una aplicación que cumple estas condiciones iniciales y se ha tratado de aportar, además, otras dos funcionalidades poco comunes en el campo de las aplicaciones de gestión de tareas.

Como se ha comentado, una tarea tiene asociada una alarma que indica al usuario la fecha de vencimiento, es decir, la fecha límite para completarla. Esa alarma puede interrumpir al usuario y demanda su atención. Esto puede ser un problema cuando el usuario tiene activos más servicios que también pueden requerir una interacción, como por ejemplo una aplicación que avise cuando se tenga cerca un cajero automático. En este ejemplo, un usuario ocupado que dispone de poco tiempo, solo debería ser interrumpido por los avisos más importantes según sus necesidades y su contexto.

Es por ello que **en esta aplicación es posible adaptar y personalizar las alarmas a las necesidades del usuario.** Así, si una alarma pertenece a una tarea de prioridad baja, su aviso será más discreto y distinto que, por ejemplo, una alarma de una tarea

de prioridad máxima. Esta gestión de lo que se llama “nivel de molestia” es una de las aportaciones principales del proyecto.

Por otra parte, **la segunda característica principal de nuestra aplicación es su integración con una red social**. Una red social es una estructura social compuesta por grupos de personas conectadas por distintos tipos de relaciones. En internet, Facebook es el principal y más extendido sitio web donde se puede participar en redes sociales. Es por ello que se ha decidido incluir e importar, en la lista de tareas, los eventos que un usuario de Facebook pueda tener vigentes.

1.3. Estructura del Proyecto

El resto del documento se organiza de la siguiente forma:

- El capítulo 2 introduce el contexto tecnológico sobre el cual se basa la aplicación de gestión de tareas, los motivos por los que se ha desarrollado el proyecto bajo la plataforma Android, y sus principales características.
- El capítulo 3 establece un marco descriptivo y comparativo entre las aplicaciones gratuitas ya existentes, las cuales gestionan listas de tareas.
- El capítulo 4 describe la propuesta completa para la gestión de listas de tareas. Muestra además las dos aportaciones citadas en esta sección, describe la estructura de datos que se ha diseñado para extraer e insertar datos en la base de datos, la gestión de alarmas y el flujo de acciones posible.
- El capítulo 5 sintetiza las conclusiones a las que se ha llegado tras la implementación y uso de la aplicación. Además, este capítulo resume posibles futuros trabajos.

Capítulo 2. Contexto Tecnológico

Una plataforma de desarrollo es el entorno de software común en el cual se desenvuelve la programación de un grupo definido de aplicaciones. Suele estar ligada a un sistema operativo, y también a lenguajes de programación o a una interfaz de programación de aplicaciones (API por sus siglas en inglés). [4]

Para la realización de este proyecto se ha elegido a Android como plataforma software de desarrollo. Uno de los principales motivos de esta elección es la limitación hardware que requiere el sistema operativo de Apple. Es necesaria una licencia para programar iOS, además de dispositivos móviles concretos de la misma marca, como por ejemplo MacBook, o iPhone. Sin embargo, Android tiene licencia libre de código abierto que permite realizar pruebas del software en el propio dispositivo sin disponer de licencia de desarrollador, como la que necesita en Apple iOS.

2.1. Android

Android es un sistema operativo basado en la plataforma software de Linux, sobre kernel 2.6, desarrollado inicialmente para dispositivos móviles por Android Inc. en Palo Alto, California, antes del año 2005. En Julio de 2005, Google compra esta empresa y sigue con su desarrollo y diseño. Poco después, en noviembre, se une al Open Handset Alliance [7], una alianza comercial de 78 compañías para desarrollar estándares abiertos para dispositivos móviles. Sus miembros destacados son Htc, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile, Nvidia y Wind River Systems. Su principal producto es Android.

Tal como podemos ver en la Figura 1, según la consultora Needham & Company [14], la cuota de mercado mundial en el primer cuatrimestre del 2011 sitúa líder a Android por encima del 35%, después de un crecimiento constante en tan solo dos años y partiendo de una cuota de menos del 2% a finales del 2008. Desde el lanzamiento de Android puede verse como su competencia más directa sigue una tendencia a la baja, a excepción de iPhone que se acerca paulatinamente a una cuota de mercado del 20%.

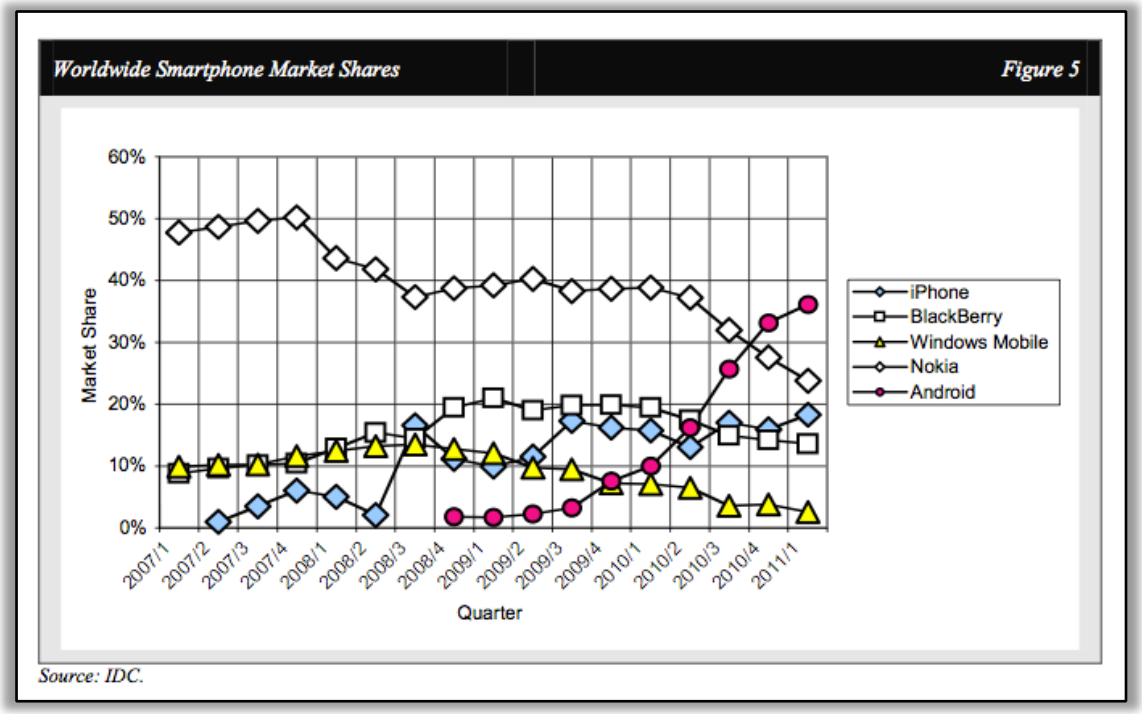

Figura 1 - Cuota de mercado mundial en Smartphones

Respecto a las distribuciones software, las cuales son compilaciones de software específico, ya compiladas y configuradas [5], la más utilizada se establece oficialmente, a fecha de Julio de 2011, en la versión Android 2.2 con casi un 60%, la cual tiene un nivel de API 8 [9].

Le siguen con aproximadamente un 18% cada una las versiones Android 2.1 con un nivel de API 7, y Android 2.3.3 / 2.3.4 con un nivel de API 10.

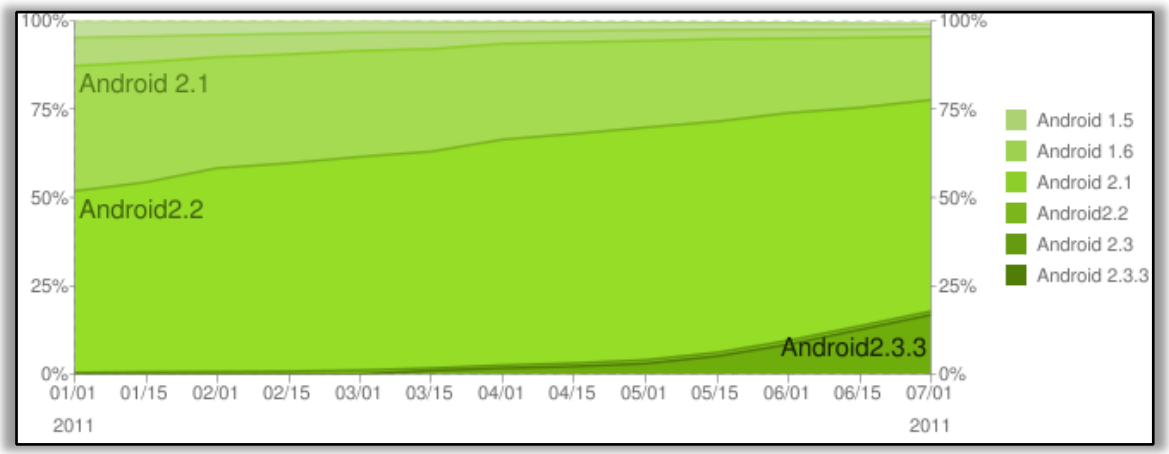

Figura 2 - Versiones de Android utilizadas en Julio 2011

2.1.2. Características Actuales de la Versión 3.0

A continuación se muestran las principales características y especificaciones de la plataforma Android que se han ido incrementando hasta la versión 3.0.

- Plataforma adaptable a todo tipo de pantallas, biblioteca de gráficos 2D, y 3D basada en OpenGL ES 2.0.
- Almacenamiento de datos con la base de datos SQLite.
- Conectividades GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE y WiMAX.
- Mensajería SMS y MMS, además de Android Cloud to Device Messaging Framework (C2DM) disponible desde Android 2.1 como servicio de Push Messaging.
- Navegador web basado en el motor de renderizado WebKit, emparejado con el motor JavaScript V8 de Google Chrome.
- No soporta J2ME ya que no hay Máquina Virtual de Java en la plataforma. Sin embargo puede ser agregado con aplicaciones de terceros como el J2ME MIDP Runner. El código que se ejecuta en Android corre en una Máquina Virtual específica de Android llamada Dalvik.
- Soporta los formatos multimedia WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), ACC, HE-ACC (en contenedores MP4 o 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP.
- Soporta streaming RTP/RTSP (3GPP PSS, ISMA) , descarga progresiva de HTML (HTML5 <video> tag). Adobe Flash Streaming (RTMP) mediante Adobe Flash Player.
- Soporta hardware como cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad, de presión, termómetro, aceleración 2D y 3D.
- El entorno de desarrollo integrado es Eclipse (actualmente 3.4 o 3.5) usando el plug-in de Herramientas de Desarrollo de Android.
- Android Market. Es una tienda de software en línea para descargar e instalar aplicaciones directamente en el dispositivo Android.
- Soporte nativo para pantallas multi-táctiles.
- Soporte para A2DF y AVRCP desde la versión 1.5. Envío de archivos (OPP) y la exploración del directorio telefónico desde la versión 2.0, y marcado de voz y envío de contactos entre teléfonos desde la versión 2.2.
- La versión principal de Android no soporta videollamada.
- Multitarea real de aplicaciones.
- Búsqueda en Google a través de la voz.
- Soporta tethering desde la versión 2.2 para utilizar el teléfono como un punto de acceso alámbrico o inalámbrico. Para una versión inferior se debe utilizar software de terceros como PdaNet.

2.1.3. Arquitectura

Los componentes principales del sistema operativo de Android son [8]:

- Aplicaciones. Están escritas en el lenguaje de programación Java. Se incluyen programas base como correo electrónico, contactos, navegador, etc…, así como aplicaciones de terceros.
- Marco de trabajo de aplicaciones. Los APIs del framework utilizados por las aplicaciones base están disponibles para los desarrolladores. La arquitectura está diseñada para simplificar la reutilización de componentes, sujeto a reglas de seguridad.
- Bibliotecas. Corriendo sobre el kernel, Android incluye un conjunto de librerías en C/C++. Algunas son librerías gráficas que incluyen SGL y OpenGL para gráficos 2D y 3D, SQLite para soporte a bases de datos, entre otras.
- Runtime de Android. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik.
- Núcleo Linux. Android depende del kernel 2.6 de Linux para los servicios base del sistema como seguridad, gestión de memoria, o gestión de procesos. El núcleo actúa también como una capa de abstracción entre el hardware y el software.

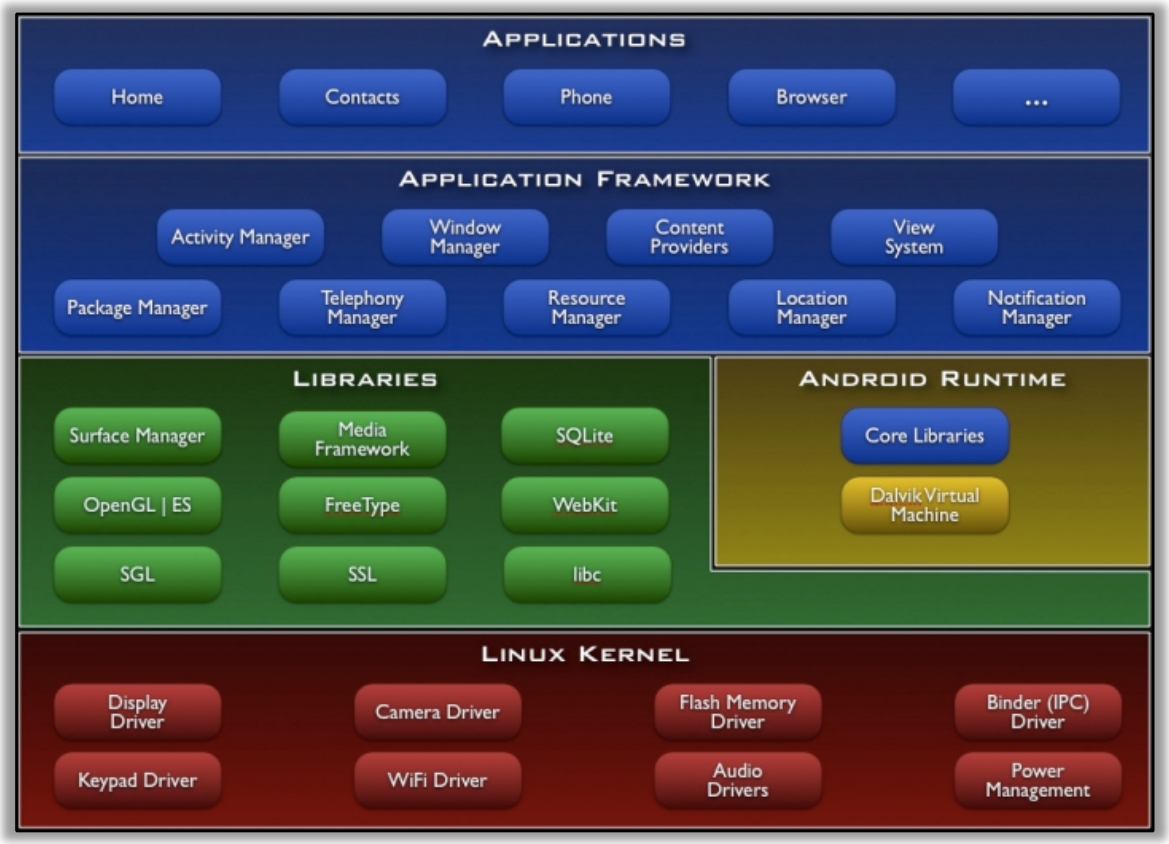

Figura 3 - Diagrama de componentes del sistema operativo de Android

2.1.4. Componentes

Una característica principal de Android es que una aplicación puede hacer uso de elementos de otras aplicaciones, siempre que éstas lo permitan. Por ejemplo si una aplicación necesita una función que ya está implementada en otra, no es necesario reescribirla, ni enlazarla, simplemente se puede ejecutar una porción de la aplicación que sea necesaria. A diferencia de otros sistemas, las aplicaciones de Android no tienen sólo un punto de entrada, sino que se pueden instanciar y ejecutar sus componentes según sea necesario.

Existen cuatro tipos de componentes cuyos objetivos y ciclos de vida difieren.

- Componente de tipo Activity

  Una actividad presenta una interfaz de usuario visual única e independiente a las demás actividades. Cada una es implementada como una subclase de la clase base Activity. Así pues, una aplicación puede consistir en una actividad tan solo, o puede contener muchas.

  Cada actividad ofrece una ventana por defecto a pintar. Se puede incluso hacer uso de ventanas adicionales, tales como diálogos en ventanas emergentes o avisos en ventanas de información. El contenido visual de una ventana proviene de una jerarquía de vistas heredadas de la clase View. Android tiene numerosas vistas listas para ser usadas, como por ejemplo botones, campos de texto o casillas de verificación.

- Componente de tipo Service

  Un servicio no tiene una interfaz de usuario visual, ya que por el contrario es ejecutado en un segundo plano por un periodo de tiempo indefinido. Cada servicio extiende la clase base Service. Por ejemplo si una aplicación reproduce música, dicha música no será manejada por actividades sino que se reproduce mediante un servicio en un segundo plano, ya que es de esperar que el usuario quiera realizar otras acciones mientras esto sucede.

  Es posible conectar con un servicio que se esté ejecutando, o ejecutarlo si todavía no lo está. Una vez conectado es posible comunicarse mediante una interfaz, y así, en el ejemplo del reproductor de música, permitir al usuario pausar, parar, avanzar, u otras acciones necesarias.

  Como las actividades y otros componentes, los servicios se ejecutan en la tarea principal de la aplicación, así pues no bloquean otros componentes o la interfaz de usuario, sino que a menudo generan otros hilos de ejecución para tareas que consuman tiempo.

- Componentes de tipo Broadcast Receiver

  Un receptor de emisiones, llamado Broadcast Receiver, es un componente que únicamente recibe y reacciona a anuncios emitidos. Por ejemplo, anuncios sobre un cambio en la zona horaria o avisos de batería baja. Las aplicaciones pueden, por lo tanto, iniciar componentes de tipo Broadcast Receiver para anunciar lo necesario a otras aplicaciones.

  Un Broadcast Receiver no muestra una interfaz de usuario, pero puede empezar una actividad en respuesta a un anuncio, usar el gestor de notificaciones, el vibrador del dispositivo, generar sonidos, o utilizar la luz.

- Componentes de tipo Content Provider

  Un proveedor de contenido, o Content Provider establece que los datos de una aplicación estén disponibles para otras aplicaciones. Esta información puede estar almacenada en el archivo del sistema, en una base de datos SQLite, o en cualquier otra forma que tenga sentido.

  Las aplicaciones no llaman directamente a los métodos que permiten recuperar o almacenar datos, sino que crean un objeto de tipo Content Resolver y utilizan sus métodos para comunicarse con el Content Provider.

2.1.5. Activación de Componentes Mediante Intent Filter

Tres de los cuatro tipos de componentes (Activity, Service, y Broadcast Receiver), son activados por un mensaje asíncrono llamado Intent. Estos Intent enlazan componentes individuales con otros en tiempo de ejecución.

Para actividades y servicios, un Intent define la acción a realizar y puede especificar la referencia de los datos a tratar con un identificador uniforme de recurso, también llamado URI, del inglés Uniform Resource Identifier. Por ejemplo, puede ser necesario empezar una actividad con el propósito de recibir un resultado, en cuyo caso, la actividad además devuelve el resultado en un Intent. Para un componente de tipo Broadcast Receiver, el Intent simplemente define el anuncio de su emisión.

El último tipo de componente, el Content Provider no es activado mediante Intent, sino cuando es objetivo de una solicitud de un ContentResolver. Se crea así, por seguridad, una capa de abstracción entre el ContentProvider y el componente que requiere información.

Hay distintos métodos para activar cada tipo de componente:

- Se puede empezar una actividad pasando un Intent a startActivity() o startActivityForResult() si se quiere que la actividad devuelva un resultado.
- Se puede empezar un servicio pasando un Intent a startService().
- Se puede iniciar un Broadcast Receiver pasando un Intent a métodos tales como sendBroadcast(), sendOrderedBroadcast() o sendStickyBroadcast().
- Se puede modificar una consulta a un Content Provider llamando a query() en un Content Resolver.

2.1.6. Fichero Android Manifest

Antes de que el sistema Android pueda ejecutar un componente de aplicación, debe conocer que el componente existe leyendo el archivo de manifiesto AndroidManifest.xml. Además de declarar componentes este manifiesto hace numerosas cosas, tales como:

- Identificar permisos de usuario, como por ejemplo, acceso a internet o acceso de lectura a los contactos del usuario.
- Declarar el nivel de API mínimo exigido para la aplicación, basado en qué API utiliza la aplicación. Esto está ligado a la versión de Android instalada. Por ejemplo, el dispositivo HTC Tattoo, cuya versión instalada es Android 1.6, soporta como máximo el API de nivel 4.
- Declarar características de software y hardware usadas o requeridas por la aplicación, tales como la cámara, servicios de Bluetooth o pantalla multitáctil.
- Y más.

2.1.7. Declaración de Componentes

La tarea primaria del archivo Android Manifest es informar al sistema acerca de los componentes de la aplicación. Por ejemplo, un archivo Android Manifest puede declarar una actividad como sigue:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

Ejemplo de AndroidManifest.xml

En el elemento <application>, el atributo android:icon apunta a los recursos por un icono que identifica la aplicación.

En el elemento <activity>, el atributo android:name especifica el nombre de clase completo de la subclase de Activity y el atributo android:label especifica una cadena a usar como etiqueta visible para el usuario de la actividad.

Se pueden declarar todos los componentes de la aplicación de este modo:

- Elementos <activity> para componentes de tipo Activity.
- Elementos <service> para componentes de tipo Service.
- Elementos <receiver> para componentes de tipo Broadcast Receiver.
- Elementos <provider> para componentes de tipo Content Provider.

Los componentes de tipo Activity, Service y Content Provider que se incluyan en el código fuente, pero no se declaren en el manifiesto, no son visibles por el sistema, y en consecuencia no podrán ser ejecutados.

Sin embargo, los componentes de tipo Broadcast Receiver pueden ser declarados en el manifiesto o creados de forma dinámica en el código (como objetos de tipo BroadcastReceiver) y registrados con el sistema con la llamada a registerReceiver().

2.1.8. Declaración de las Capacidades de Componentes

Tal como ya se ha comentado, se puede utilizar un objeto de tipo Intent para empezar componentes de tipo Activity, Service y Broadcast Receiver. Se puede hacer de forma explícita en el Intent, nombrando el componente como objetivo usando el nombre de clase del componente. Sin embargo, el poder real de los objetos de tipo Intent reside en el concepto de "*Intent Action*". Con Intent Action simplemente se describe el tipo de acción que se desea realizar, y opcionalmente, los datos en que se desea realizar la acción. Además, permite al sistema encontrar un componente en el dispositivo que pueda realizar la acción y comenzarla. Si hay múltiples componentes que realicen esa misma acción descrita por Intent, entonces el usuario selecciona cuál de ellos usar.

La manera en la que el sistema identifica los componentes que pueden responder a un Intent es comparando el Intent recibido con los elementos de tipo Intent Filter previstos en el archivo Android Manifest de otras aplicaciones del dispositivo.

Cuando se declara un componente en el archivo Android Manifest, se puede opcionalmente incluir elementos de tipo Intent Filter que declaren las capacidades del componente que puedan responder a otros objetos Intent de otras aplicaciones. Se puede declarar un Intent Filter para un componente, añadiendo el elemento <intent-filter> como hijo del elemento de declaración del componente.

Por ejemplo, una aplicación de correo electrónico con un componente de tipo Activity, para crear un nuevo email, puede declarar un elemento Intent Filter en su archivo Android Manifest para responder a objetos Intent de “envío” (para enviar emails). Un componente Activity en otra aplicación puede entonces crear un objeto Intent con la acción “envío” (ACTION_SEND), la cual encaje con la actividad “envío” de la aplicación de correo y sea lanzada cuando se invoque el objeto Intent con el método startActivity().

2.1.9. Declaración de Requisitos de la aplicación

Existe una gran variedad de dispositivos Android, y no todos poseen las mismas características ni capacidades. Para prevenir que una aplicación se instale en dispositivos con carencias requeridas por esa aplicación, es importante que se defina un perfil claro para el tipo de dispositivos que esa aplicación soporta declarando en el archivo de manifiesto requisitos de dispositivos y software. Muchas de esas declaraciones son únicamente informativas y el sistema no las lee, pero servicios externos como Android Market hace lecturas de ellas para proveer un filtro a los usuarios cuando busquen aplicaciones desde su dispositivo.

Por ejemplo, si una aplicación requiere una cámara y usa una API introducida en Android 2.1 (Nivel de API 7), se debería declarar esto como requisitos en el archivo manifiesto. De este modo, los dispositivos que no tengan cámara y tengan una versión Android más baja de 2.1 no puedan instalar esa aplicación desde Android Market.

Sin embargo, se puede declarar que una aplicación use la cámara, pero no lo requiera. En este caso, la aplicación debe disponer de un test en tiempo de ejecución para determinar si el dispositivo tiene cámara y deshabilitar cualquier característica que use la cámara si no está disponible.

Algunas de las más importantes características de dispositivo que se deban considerar en una aplicación son:

- Tamaño y densidad de pantalla

  Android define dos características para cada dispositivo, las dimensiones físicas de la pantalla, y la densidad física de píxeles en la pantalla, o puntos por pulgada. Para simplificar los tipos de configuraciones de pantallas se han creado grupos.

  Los tamaños de pantalla son: pequeño, normal, grande y extra grande.

  Las densidades de pantalla son: densidad baja, densidad media, densidad alta, y densidad extra alta.

  Por defecto una aplicación es compatible con todos los tamaños y densidades, ya que Android ajusta de forma apropiada las capas de la interfaz

de usuario y los recursos de imágenes. Sin embargo, se pueden crear capas específicas para ciertos tamaños, e imágenes para ciertas densidades utilizando recursos alternativos, declarando en el fichero Android Manifest qué tamaños de pantalla de la aplicación lo soporta con el elemento <supports-screen>.

- Configuraciones de entrada

  Muchos dispositivos disponen de diferentes mecanismos de entrada, tales como teclados hardware o ruedas de desplazamiento. Si se requiere un tipo particular de hardware, es posible declarar en el archivo Android Manifest el elemento <uses-configuration>. Aunque es raro utilizarlo.

- Características del dispositivo

  Muchas características hardware y software de un dispositivo Android pueden existir, o no. Tales como la cámara, el sensor de luz o el Bluetooth. No se debe asumir con certeza que una característica esté disponible, así que se debe declarar cada característica que requiera la aplicación en el elemento <uses-feature>.

- Versión de la plataforma

  Diferentes dispositivos Android a menudo ejecutan diferentes versiones de la plataforma Android, tales como Android 1.6 o Android 2.3. Cada versión sucesiva incluye API adicional no disponible en la versión anterior. Se debe declarar por tanto la versión mínima del nivel de API utilizando el elemento <uses-sdk>.

2.1.10. Recursos de la Aplicación

En una aplicación Android se requieren recursos además del código, tales como imágenes, archivos de sonido, y cualquier cosa relacionada con la interfaz visual.

Por cada recurso que se incluya en el proyecto, la herramienta SDK define un único entero como ID, con el que se puede referenciar el recurso desde el código de la aplicación. Por ejemplo si la aplicación contiene una imagen llamada logo.png (guardada en el directorio res/drawable/), la herramienta SDK generará un ID del recurso llamado R.drawable.logo, que puede usarse para referenciar e insertarlo en la interfaz de usuario.

Uno de los aspectos importantes de proveer recursos separados del código fuente es la posibilidad de utilizarlos según convenga para diferentes configuraciones del dispositivo. Por ejemplo definiendo cadenas de la interfaz gráfica en XML, se pueden traducir en otros idiomas y guardar esas cadenas en distintos archivos. Entonces en base al idioma que se especifica en el directorio del recurso (tal como

res/values-fr/ para cadenas con valores en Francés) y la configuración del idioma del usuario, el sistema Android aplica el idioma apropiado para la interfaz de usuario.

2.2. Conclusiones

En este capítulo se ha descrito la plataforma de desarrollo software Android para el desarrollo de aplicaciones móviles.

El proyecto que se presenta es un desarrollo de una aplicación móvil para la gestión de tareas personales bajo la plataforma Android. Ya que Android y iOS son los sistemas operativos para Smartphone más extendidos mostramos brevemente por qué nos hemos decantado por Android.

Una de las principales ventajas es que el código es abierto y libre. Google liberó Android bajo licencia Apache 2.0 [10], lo que significa que cualquier programador puede realizar una aplicación para esta plataforma. En cambio iOS necesita una licencia específica que debe ser pagada y aprobada por Apple. [11]

En cuanto al lenguaje de programación utilizado, Android utiliza Java en su Kit de Desarrollo Software y iOS Objective-C [12] que actualmente se usa como lenguaje principal de programación en Mac OS X. Esta característica nos ha llevado a elegir Android, ya que nos permite trasladar mejor los conocimientos y lenguajes de programación aprendidos a lo largo de la carrera en este proyecto.

A partir de la información obtenida y como se puede ver en la anterior Figura 2, la versión Android más extendida a fecha de Julio de 2011 es la 2.2. Este es uno de los motivos por los que nuestra aplicación ha sido programada bajo la plataforma Android 2.2 con un nivel de API 8.

Sin embargo, el principal motivo ha sido la necesidad de utilizar un componente externo del tipo CommonsWare Android Componentes, o CWAC. Estos componentes son librerías de código abierto que ayudan a solucionar algunos problemas tácticos en el desarrollo con Android.

El componente CWAC que se ha utilizado es cwac-wakeful, que proporciona el servicio WakefullIntentService, una mejora del servicio IntentService que mantiene el dispositivo despierto mientras las tareas se siguen procesando en segundo plano. Nos resulta útil para las alarmas programadas a través de objetos de la clase AlarmManager.

Un requisito de la versión utilizada de este componente exige la versión de Android 2.0 o superior [15], ya que así se tiene la ventaja de utilizar su método onStartCommand() para un mejor manejo de servicios erróneos.

Capítulo 3. Análisis de Aplicaciones Móviles para la Gestión de Tareas

Para establecer un marco común entre la aplicación de gestión de tareas presentada y las existentes en Android Market se han analizado versiones gratuitas de aplicaciones de tareas cuya versión mínima fuera preferentemente Android 2.2. La búsqueda se ha realizado mediantes las palabras claves "TODO", "TODO list", "tareas", y "lista de tareas" además de las búsquedas relacionadas que ofrece la web de Android Market.

En particular, las aplicaciones que se han considerado interesantes a analizar, debido a su número de descargas y a su popularidad, son: Checkmark, Gtasks, Jorte, Schedule st, Tareas Astrid, Task List, Task To Do Free, Todo Task Manager, y TASKOS.

Este capítulo realiza un análisis descriptivo de cada una de estas aplicaciones, y finalmente, muestra una tabla comparativa que permite ver de una forma clara las características, similitudes y diferencias entre estas aplicaciones.

3.1. Aplicaciones

Checkmark

Uno de los aspectos más relevantes de la aplicación Checkmark [29] es la posibilidad de crear tareas además de listas de verificación, también conocidas como listas de la compra. Las tareas pueden configurarse con un tiempo de inicio, y un tiempo de vencimiento, y sólo los ítems de la lista de verificación tienen un tiempo de vencimiento sin hora.

Es posible establecer un orden según fecha de vencimiento, categorías, títulos, prioridades, etc. Y filtrar la lista para mostrar u ocultar elementos.

Ofrece la opción de exportar de forma individual cada tarea o lista de verificación, por lo menos en esta versión gratuita y a la hora de importar también debe hacerse de uno en uno. Al compartir un elemento vía e-mail existe la misma limitación.

Ya que no dispone de calendario, existe una segunda aplicación llamada "Checkmark Calendar" que en combinación con "Checkmark" permite mostrar las tareas y listas de verificación en varias vistas de calendario, e integrarlo con eventos sincronizados con Google Calendar.

En cuanto a la interfaz gráfica de Checkmark, como podemos ver en la Figura 4, destaca la gama de colores utilizada inusual en el diseño general de Android. Sus fondos imitan la textura de papel desgastado y los colores desentonan con degradados

azules y grises amarronados. Cabe destacar que puede configurarse el tamaño de fuente, y el máximo número de líneas de una nota.

En el dispositivo móvil HTC Tattoo con Android 1.6 se cierra inesperadamente con bastante frecuencia aunque en sus requisitos se especifica que la versión mínima de Android es 1.5 o superior.

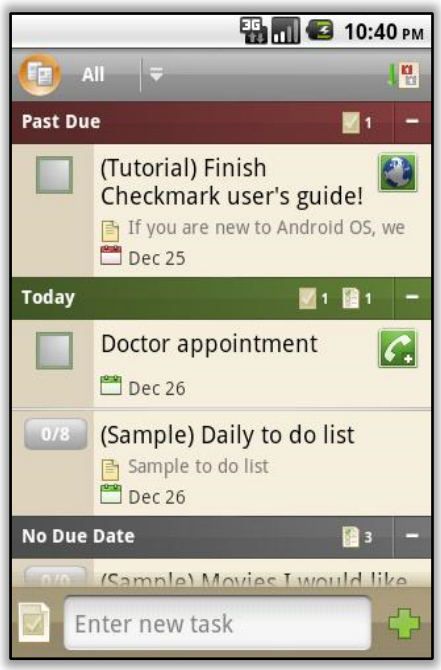

Figura 4 - Aplicación Checkmark

GTasks

La aplicación Gtasks [30] dispone de una interfaz muy sencilla, con prioridad en las categorías llamadas aquí listas, ya que se navega en la pantalla principal entre ellas deslizándolas a izquierda o derecha.

Al añadir una tarea se muestra un aspecto de nota con líneas y fondo amarillo, con una fecha de vencimiento que integra una alarma obligatoria, con posibilidad de repetición.

Las opciones para compartir uno o todos los elementos de la lista son amplias, ya que permite enviar por numerosas aplicaciones como Correo, Facebook, Mensajes, Peep (Twitter), etc.

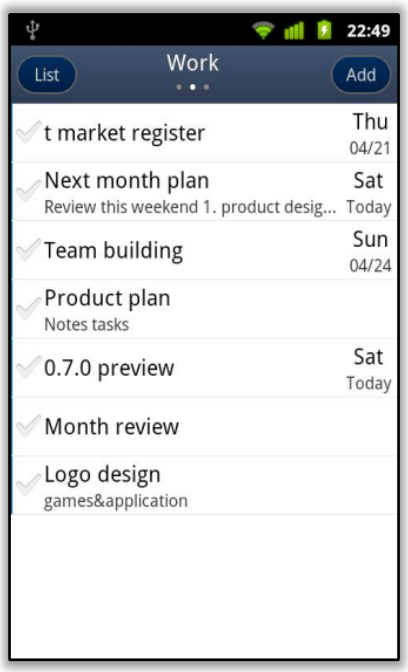

Figura 5 - Aplicación GTasks

Jorte

En esta aplicación se integran eventos con tareas, y se muestran ambos en una vista principal de calendario o sólo las tareas en una lista de tareas aparte. Por ejemplo es posible tener programado un evento el día 13 de julio, como puede ser un cumpleaños, y además en ese mismo día haber incluido una tarea que deba realizarse como por ejemplo comprar una tarta en una pastelería específica. El evento quedaría marcado en el calendario, y la tarea en el listado de tareas.

Las tareas están en un segundo plano, se describen también como notas, y sólo disponen de dos prioridades, importante y no importante, además de la necesaria fecha de vencimiento, que en este caso no cuenta con alarma ni avisos. Se muestra en rojo si ha vencido la fecha, o en gris si se ha completado.

Además Jorte [31] trata de facilitar el uso con botones directos situados en la esquina inferior derecha. Aunque se alternan sus iconos y no se intuye la acción que tienen asociada. Estas acciones son varias, y son rotatorias o secuenciales ya que desde un solo botón se va cambiando entre ellas.

El izquierdo muestra u oculta una lista inferior en el calendario de eventos, o de tareas, o de ambos, y con el botón derecho es posible alternar varias vistas de calendario, mensual, de una semana, de dos semanas, modo agenda, etc…

En cuanto a la sincronización, es configurable con Google Calendar y con Google Tasks, aunque es una aplicación enfocada principalmente a eventos, con el

añadido de poder usar tareas sencillas, ya que por ejemplo no es posible editar las categorías o listas de tareas. Ofrece la opción de importar o exportar eventos, o tareas.

Por su parte, los eventos cuentan con un interesante campo de localización, que enlaza directamente en la búsqueda de Google Maps.

Figura 6 - Aplicación Jorte

Schedule st

Esta aplicación con aspecto de calendario limita su funcionalidad a eventos, no siendo posibles las tareas ni las notas como describen los autores. Siendo la traducción TODO List como “tareas pendientes” queda en evidencia que los eventos no son tareas y esta funcionalidad está de más, ya que no existe.

Aunque ya dispone de una interfaz sencilla y clara, mediante un botón inferior se accede a un listado de plantillas que pueden comprarse para sustituir la original y así cambiar la apariencia.

La interfaz Schedule st [32] es similar a Jorte, por ejemplo en el menú que aparece cuando se hace un largo clic en un día del calendario, para cambiar el color del día o mostrar u ocultar la festividad.

Añade también la localización del evento con el enlace a búsqueda de Google Maps como ya dispone Jorte. Destacamos también su sincronización con Google Calendar.

Figura 7 - Aplicación Schedule st

Tareas ASTRID

Astrid [33] tiene un largo recorrido como aplicación, con un progreso relevante que le ha convertido en la aplicación más usada, en torno a las 5.000.000 de instalaciones.

Tiene una interfaz que trata de aprovechar al máximo el espacio con elementos muy reducidos, las prioridades se muestran como líneas verticales de colores que ocupan muy poco espacio, y las letras de la segunda línea de información tienen un tamaño de fuente mínimo. Tan sólo la primera línea, correspondiente al título, es modificable de tamaño.

Un aspecto relevante es la temporización en el seguimiento de tareas. Así pues, un cronómetro nos avisa del tiempo empleado en dicha tarea, pudiendo pausarlo o retomarlo.

No dispone de listas ni categorías, ya que son sustituidas por un sistema de etiquetado que clasifica y complementa las tareas. Se pueden añadir varias etiquetas a una misma tarea. Este campo de etiquetado dispone de autocompletado con las etiquetas que ya han sido insertadas con anterioridad. También es posible la ordenación o filtrado por etiquetas.

Destaca frente a otras aplicaciones el filtrado personalizado, pudiendo añadir y combinar criterios para crear un filtro nuevo.

Astrid dispone de componentes adicionales que se pueden añadir a la aplicación principal, por ejemplo Astrid Locale Plugin, que integra Google Locale, y Astrid Power Pack que añade widgets e integra funcionalidades de voz.

Sincroniza con Google Tasks y gestiona las copias de seguridad que hace de forma automática.

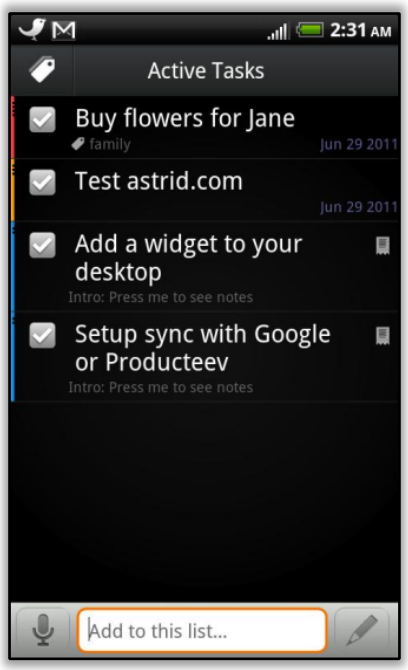

Figura 8 - Aplicación Tareas ASTRID

Task List

Como podemos ver en la Figura 9, Task List [34] es una de las pocas aplicaciones que cuenta con una guía inicial de aprendizaje que nos muestra de forma visual su funcionalidad.

Task List es además la aplicación que más opciones de configuración permite en comparación con las demás, pudiendo cambiar la apariencia de las listas que se muestran como pestañas, ordenación, formatos de fecha, opciones de edición de tareas, etc.,…

La funcionalidad más interesante es la configuración de notificaciones. Excepto la elección del color del LED del dispositivo, se pueden configurar las notificaciones según tres prioridades ya establecidas: baja, media y alta. Así que es posible cambiar para cada una de ellas el tono o melodía que sonará, y el tipo de aviso según “sonido normal”, “sonido normal y vibración”, “sonido intenso”, “sonido intenso y vibración”. En realidad las aplicaciones disponen de cuatro niveles de prioridad, habiendo dos

tipos de prioridad media, en color amarillo y naranja. Lo cual no se corresponde con la configuración de las tres prioridades comentadas.

Estas alarmas avisan al usuario con una barra de notificación, cuyo clic permite posponer el aviso, o dar la tarea como completada.

Task List hace también un seguimiento de las tareas según su fecha de vencimiento, cambiando de color el elemento. Se muestra en rojo cuando la tarea ha vencido, amarillo si vence el mismo día, y gris si la tarea se ha completado.

A pesar de no ofrecer ningún tipo de sincronización permite exportar e importar tareas a un archivo propio de la aplicación de tipo XML en la tarjeta de memoria SD. Este archivo se llama TaskList.xml y no es posible renombrarlo, tan solo se sobrescribe al exportar tareas. Por lo tanto se cuenta con un solo fichero de respaldo.

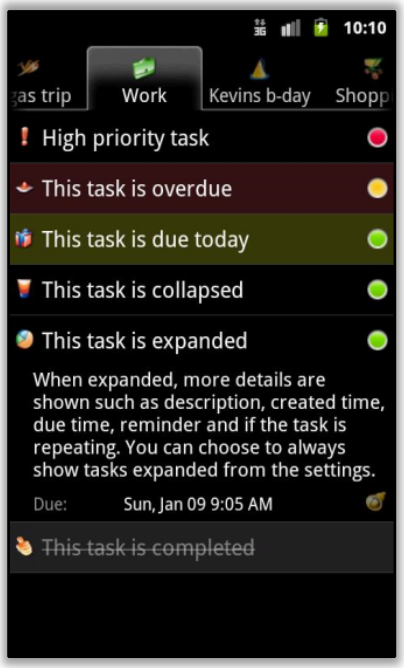

Figura 9 - Aplicación Task List

Task To Do Free

Uno de los aspectos más relevantes de Task To Do Free [35] es el seguimiento de tareas que se realiza con unos botones laterales similares a los de un reproductor de música. Ofrece las opciones “not started”, “started”, “waiting” y “done”.

Además en la parte inferior derecha hay tres botones. Uno nos permite ordenar las tareas según el seguimiento anterior, o bien por fecha de vencimiento. El central permite mostrar u ocultar tareas por su categoría. El derecho, por último, mostrar u ocultar tareas según el mismo filtro del primer botón.

En el listado se muestra a la derecha de la tarea la fecha de vencimiento, la cual cambia de color según se acerca, y se informa además si dispone de alarma o no para ese momento.

Dispone además de sistema de copia de seguridad que permite exportar tareas en formato XML.GZ. Se crea un archivo de este tipo en la tarjeta de memoria SD cada vez que se exportan las tareas. De este modo es posible restaurar e importar la lista de tareas desde distintos archivos de respaldo. El problema es que no es posible eliminar desde Task To Do Free estos archivos, y la lista de archivos de estas copias de seguridad puede llegar a ser muy larga.

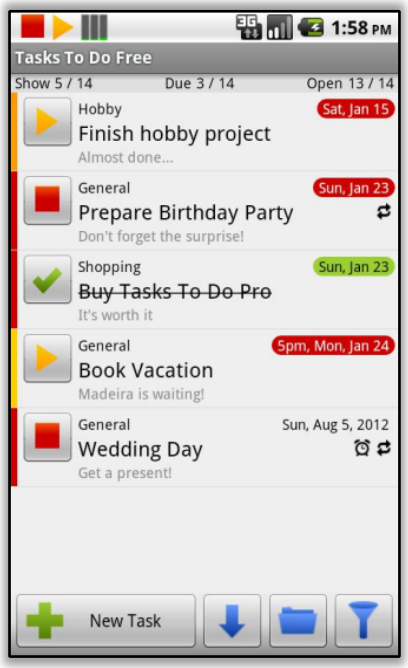

Figura 10 - Aplicación Task To Do Free

Todo Task Manager

Todo Task Manager [36] dispone de una vista avanzada para tareas que permite especificar el porcentaje completado de la tarea, o por ejemplo el número de horas que le hemos dedicado.

Su lista principal cuenta con casillas de verificación para finalizar tareas, se muestra además la categoría, una pequeña nota y una fecha que cambia de color según se acerca la hora de vencimiento.

Cabe destacar que en sus preferencias es posible configurar que se realice una copia de seguridad de forma automática. También permite cambiar el sistema de prioridad, algo novedoso que ninguna aplicación de las analizadas dispone. Estos sistemas pueden ser numéricos (1, 2, 3, 4, 5), sistema Covey (1A, 1B, 2A, 2B), Alta /

Baja / Normal, sistema Moscow (Hazlo, Debería, Quizás…), método Eisenhower, o sistema A/B/C/D/E.

Se puede personalizar también el tema de la aplicación y cambiar el estilo y color de letras, y el del fondo.

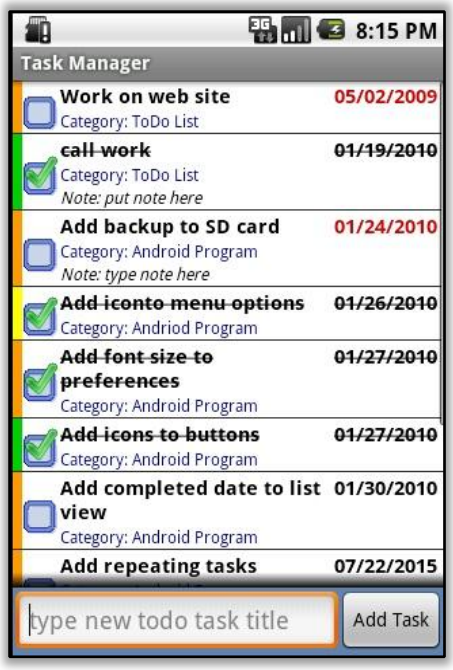

Figura 11 - Aplicación Todo Task Manager

TASKOS

La sencilla interfaz gráfica que muestra TASKOS [37], hace de ésta una aplicación usable, fácil e intuitiva. Dispone de una entrada de texto superior, para añadir rápidamente una tarea. La inserción de la tarea puede ser además mediante entrada por voz.

En cuanto se van finalizando las tareas pasan automáticamente a una lista donde se recogen todas las tareas finalizadas.

Dispone además de sincronización con Google Tasks y podemos incluso compartir una tarea a un contacto vía email o mediante mensaje de texto SMS.

Según las sesiones que se tengan abiertas en el dispositivo móvil integra también a los amigos del usuario de aplicaciones como Facebook, Gmail, Dropbox, Whatsapp, Beluga o Twitter.

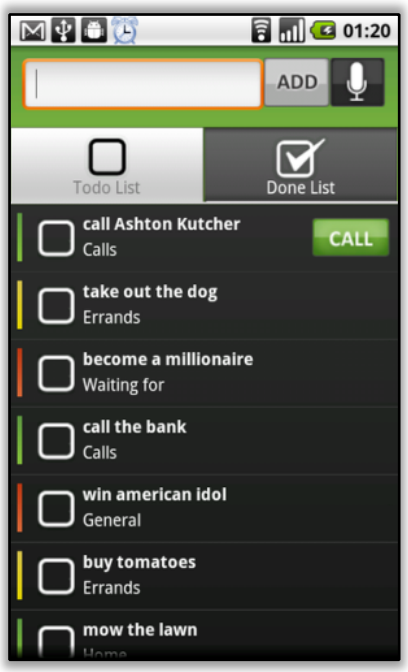

Figura 12 - Aplicación TASKOS To Do List | Task List

3.2. Tabla Comparativa

Dado que se han presentado diversas aplicaciones y todas ellas muestran características diferentes, se muestra a continuación una tabla que permite comparar las características del conjunto de aplicaciones presentado, el cual corresponde al dominio de las aplicaciones de gestión de tareas.

En particular la tabla refleja si las aplicaciones cubren las siguientes características.

Sincronización con Facebook

Una característica importante de los Smartphone es que permiten al usuario estar conectado a internet de forma permanente. Las aplicaciones tienden a incluir nuevos mecanismos de interacción. Se amplían los servicios disponibles y el usuario puede interactuar con mayor frecuencia con su entorno social.

Cuando los usuarios utilizan sus dispositivos móviles para leer noticias, ver el tiempo, el e-mail o sus redes sociales, prefieren hacerlo sobre una aplicación que directamente por la web [16]. Es por ello que existe una tendencia a integrar las redes sociales en las aplicaciones.

Se plantea entonces la cuestión sobre la integración de Facebook, como red social más extendida y usada, en este tipo de aplicaciones de gestión de tareas.

Sincronización con Google Calendar o Google Tasks

Google ofrece los servicios Calendar y Tasks para gestionar eventos y tareas respectivamente, bien en un calendario, o en un listado. La integración de estos servicios en una aplicación de gestión de tareas permite tener varias ventajas.

Una de ellas, por ejemplo, es que no se hace necesario hacer copias de seguridad de forma local en el dispositivo móvil, o de forma manual. Esta sincronización automática centraliza los eventos o tareas que se guardan en el servidor de Google. De este modo, si nuestra aplicación se desinstala, o nuestro dispositivo móvil sufre una avería, no perdemos ningún dato. Son accesibles con nuestro usuario de Google desde cualquier otro dispositivo u ordenador.

Otra ventaja derivada de la anterior es la gestión que puede hacerse de los eventos o tareas desde otro punto de acceso de forma más cómoda, como pueda ser un ordenador de sobremesa. Cuando la aplicación sincronice con Google se actualizarán los datos que se han modificados y tendremos la última versión correcta de todos los datos.

Alarmas

Cada vez más, se incrementan los servicios en los dispositivos móviles y su intromisión. Por ejemplo, en el caso de las aplicaciones de gestión de tareas, las alarmas pueden suponer una sobrecarga de avisos para un usuario que no disponga del suficiente tiempo o las notificaciones no sean importantes.

Por ello, se cuestiona que las alarmas puedan ofrecer la flexibilidad de adaptarse al usuario según sus necesidades y su contexto en términos de nivel de molestia. Este tipo de alarmas se denominan contextualizadas. Se puede tener en cuenta una variable de la tarea que la condicione. Por ejemplo, la prioridad, la fecha de vencimiento, o la ubicación del usuario puede influir en el aviso de estas tareas.

Un añadido para una alarma es su repetición, es decir, la posibilidad de insistir en el aviso de la fecha de vencimiento en la que la tarea debería haberse completado. Esta insistencia resulta útil, por ejemplo, en el caso en que un usuario no se haya dado cuenta de la alarma y necesite que se le demande más su atención.

Por último, las alarmas por voz son un mecanismo más para el aviso contextualizado de las alarmas. No es necesario en este caso disponer del dispositivo móvil a mano, y se enriquece el aviso sonoro al poder escuchar información acerca de la tarea. Estos mecanismos requieren tiempo para prestar atención al aviso sonoro, pero a su vez, no se pierde tiempo en manipular el dispositivo para recuperar información.

Widget

Un widget es un elemento visual que puede fijarse en una de las pantallas principales de Android. Una aplicación que siempre está activa y visible en el escritorio. En una aplicación de tareas es útil tener en un widget visible un listado de ellas en la pantalla principal a modo de agenda, por ejemplo.

Categorías

Cuando una lista de tareas se hace demasiado extensa es necesario poder crear categorías para poder clasificarlas y gestionarlas de una manera más cómoda. De esta forma se puede aplicar además filtros de categorías para mostrar u ocultar según convenga.

Seguimiento de Tareas

El principal objetivo en un seguimiento de tareas es el control del estado de cada tarea. Según ese estado podemos organizarnos el tiempo restante hasta la fecha de vencimiento para seguir trabajando en la tarea. Bien es posible marcar una tarea según el porcentaje de avance conseguido, o mediante grados de estado, como por ejemplo “comenzada”, “en pausa”, o “finalizada”.

	Sinc. con Google	Widget	Repeticiones de Alarma	Alarma por voz	Alarma contextualizada	Seguimiento de Tareas	Categorías	Sinc. con Facebook	Otras funcionalidades
Checkmark	V[1]	V	V	X	X	V	V	X	- Listas de casillas de verificación - Añadido "Checkmark Calendar"
Gtasks	V[2]	V	V	X	X	V	V	X	
Jorte	V[1]	V	X[3]	X	X	V	X[4]	X	- Eventos Calendario / Tareas
Schedule st	V[5]	V	V	X	X	X	X[6]	X	- Copias de seguridad - Localización manual
Tareas Astrid	V[2]	V	V	X	X[7]	V[8]	X[9]	X	- Entrada por voz - Posibilidad de añadidos - Copias de seguridad
Task List	X	V	V	X	V	V[10]	V[11]	X	
Task To Do Free	X	V	V	X	X	V[12]	V	X	- Entrada por voz - Filtrado según estado de las tareas
Todo Task Manager	X	V	V	X	X	V	V	X	
TASKOS	V[2]	V	V	X	X	X	V	X	- Entrada por voz - Compartir tareas con amigos
Aplicación Desarrollada	X	X	X	V	V	X	V	V	

Tabla 1 - Comparativa entre aplicaciones de tareas

[1] Google Calendar y Google Tasks.
[2] Google Tasks.
[3] Los Eventos de Calendario disponen de notificación. Las tareas no tienen fechas de vencimiento.
[4] La distinción de tareas se realiza con colores.
[5] Google Calendar.
[6] En lugar de categorías se utilizan distintos calendarios.
[7] Configuración individual de cada alarma.
[8] Se utiliza un cronómetro manual.
[9] Las tareas se organizan por varias etiquetas, no por categorías únicas.
[10] Cambia el color de la tarea según se acerca el vencimiento.
[11] Las categorías son organizadas como listas separadas por pestañas.
[12] Cambia el color de la fecha según se acerque el vencimiento.

3.3. Comparativa con la Aplicación Desarrollada

En la anterior tabla comparativa se han mostrado las distintas características y funcionalidades que integran las aplicaciones de gestión de tareas analizadas.

Por una parte, la integración de las redes sociales en este tipo de aplicaciones todavía no se ha comenzado a implantar. Ninguna de las aplicaciones anteriores dispone de sincronización alguna con Facebook, o cualquier otra red social. TASKOS es la única que muestra los amigos que tenemos en Facebook, para compartir con ellos una tarea concreta. Pero para que esta funcionalidad de TASKOS exista, debe haberse iniciado en Android la sesión de Facebook, por lo tanto su conexión se realiza de forma indirecta.

En cuanto a la posibilidad de realizar copias de seguridad en las aplicaciones, utilizando los servicios que ofrece Google con Google Calendar y Google Tasks, dos tercios del total de aplicaciones analizadas tienen esta opción.

Por otro lado, se ha mostrado que todas las aplicaciones disponen de repetición de alarmas, exceptuando Jorte. Esta es una opción inicial para controlar el nivel de molestia, si el usuario decide activarlo o no. En cambio, no hemos visto ningún mecanismo de control de alarmas, como la contextualización de avisos, a excepción de Task List. Las aplicaciones no dan soporte a los avisos por voz, por ejemplo con la librería TTS (Text to Speech) y no se aplica ningún criterio de distinción de avisos.

Acerca del seguimiento del trabajo completado de una tarea, en el caso de Schedule st y TASKOS no se ofrece esta mejora, y por el contrario la aplicación Tareas ASTRID resulta ser la más avanzada. Ésta cuenta con un cronómetro manual que calcula el tiempo utilizado en una tarea concreta. Otras, como Task List y Task To Do Free hacen el seguimiento proporcional automático según se acerca la fecha de seguimiento y no de forma manual como se hace con el resto de aplicaciones.

Finalmente, hemos decidido diseñar nuestra aplicación con los siguientes aspectos. Como se ha descrito, es interesante la integración de las aplicaciones con las redes sociales. La tendencia a utilizar cada vez más estos servicios sociales indica la relevancia de extender las redes sociales a las aplicaciones. A pesar de que ninguna aplicación estudiada cuenta con esta característica nos vemos obligados a cuestionarla.

Una segunda característica principal de nuestra aplicación es la capacidad de personalización de las alarmas mediante el uso de distintos mecanismos de interacción para presentarlas, otra función de la que ninguna otra aplicación en la gestión de tareas hace uso. Sin embargo, hay una tendencia a utilizar la contextualización de alarmas en cualquier tipo de aplicaciones.

Cabe destacar que Schedule st y Jorte no son aplicaciones de tareas propiamente dichas, a pesar que entre las palabras clave de su descripción en Android Market figure “TODOlist”. Es por ello que no cuentan con un sistema de categorización y ordenación de tareas como las demás. Se centran, en cambio, en calendarios, y en eventos, un concepto distinto al de tareas.

En el caso de la sincronización con los servicios Google Calendar y Google Task se ha visto la necesidad de tener los datos respaldados, y accesibles desde cualquier dispositivo. En este proyecto no se ha incluido esta funcionalidad, que pensamos importante, para no elevar la complejidad de la aplicación y acotar el trabajo a desarrollar. No se integran Google Calendar ni Google Tasks porque al hacerlo ya otras aplicaciones, se ha considerado más interesante conocer e implementar los beneficios de características que sí se han implementado como por ejemplo la gestión de alarmas o la integración con la red social Facebook.

3.4. Conclusiones

En este capítulo se han descrito y comparado características comunes de las aplicaciones más populares de gestión de tareas.

En primer lugar, se ha mostrado la tendencia de la integración de las redes sociales en los dispositivos móviles. La proliferación de nuevas funcionalidades y servicios en las aplicaciones sobre redes sociales puede justificar que también se apliquen en el ámbito de las aplicaciones de gestión de tareas.

Por otro lado, se han visto las ventajas de sincronizar los datos de las tareas con los servicios de Google Calendar y Google Tasks.

Se ha introducido el concepto de nivel de molestia de las alarmas. En ocasiones, los avisos de las alarmas pueden ser molestos para el usuario, y en otras pueden echarse en falta. Por ello, es necesaria una personalización para cada usuario donde se puedan controlar las vibraciones, los sonidos y los avisos visuales para ajustar el tipo de alarma según las necesidades del usuario.

Por último, se ha visto que el seguimiento de tareas permite al usuario un mayor control sobre el estado de las tareas, y organizarse así mejor el tiempo restante para completarlas. Se ha mostrado la importancia del uso de widgets como elemento visual para, por ejemplo, ver rápidamente la agenda de tareas a realizar. Y el uso de categorías permite una mejor organización de tareas cuando existe una gran cantidad de elementos.

Capítulo 4. Implementación de la Aplicación

En este capítulo se va a profundizar en cómo se ha diseñado e implementado la aplicación de gestión de tareas, así como los componentes de la aplicación.

Como se ha concluido en el capítulo 2.2, se pretende diseñar e implementar una aplicación bajo el sistema operativo Android. En esta plataforma software se ha hecho uso de la versión 2.2 cuya API es 8 bajo el entorno de desarrollo Eclipse y el SDK que ofrece Google.

La aplicación introduce la personalización de las alarmas mediante el control del nivel de molestia. Una característica necesaria por la tendencia al incremento de los servicios en los dispositivos móviles. Se incrementa además la cantidad de intromisión hacia el usuario que hacen estos servicios. Por ejemplo, el uso de alarmas de varias aplicaciones puede suponer un excesivo reclamo de atención del usuario que un usuario ocupado no se pueda permitir.

Por otro lado, se introduce la integración de los eventos de Facebook en la aplicación de gestión de tareas. Se plantea la necesidad de esta integración en este tipo de tareas ya que los eventos también forman parte de agenda, actividades y eventos del usuario.

Para la familiarización con el entorno de la plataforma Android se ha procedido a seguir cada uno de los tutoriales de la serie *Hello, Views* [28] de la web de desarrolladores de Android. Esta inicialización ha sido necesaria y útil, puesto que se ha comprendido el modo de diseñar e implementar vistas lineales y relativas, además de vistas en tablas y cuadriculas. Se ha aprendido a crear vistas por pestañas y vistas de elementos listados. Por último se ha conocido el modo de programar selectores de hora y de fecha, a aplicar botones personalizados, campos de texto editables, casillas de verificación, botones de opción, botones de activación y barras de puntuación. Se finalizaron los tutoriales añadiendo localizaciones en un mapa de Google Maps, y aplicando el componente visual Spinner, que permite seleccionar un elemento de una lista desplegable y deslizable.

Por otro lado, se ha estudiado la interfaz de programación de aplicaciones o API (del inglés *Application Programming Interface*) de Facebook para su aplicación en el proyecto. Se ha integrado Facebook en Android utilizando los eventos de esta red social como tareas, leyendo únicamente su fecha de vencimiento, y no la de inicio. Aquí reside la distinción entre evento y tarea. Según la Real Academia Española una tarea [38] es un trabajo que debe hacerse en tiempo limitado, y sin embargo un evento [39] es un suceso importante y programado, de índole social, académica,

artística o deportiva. Ambos tienen fecha de vencimiento, pero una tarea no debe tener necesariamente una fecha de inicio distinta a la actual.

Una vez se han tenido estos conceptos básicos de Android y Facebook asimilados se ha decidido programar la aplicación siguiendo el estilo de programación por capas [19]. El diseño que utilizaremos es el de tres niveles, que además es el más utilizado.

Un primer nivel es la capa de presentación, para recibir las solicitudes del usuario y visualizar los resultados. Este nivel es el que presenta la información al usuario en un formato correcto y captura información del usuario. Es llamada también interfaz gráfica y sólo comunica con la capa de negocio. Esta segunda capa de negocio es donde se reciben y se envían las peticiones y respuestas al usuario. Esta capa se llama también lógica del negocio porque establece las reglas a seguir y comunica tanto con la capa de presentación, para recibir las solicitudes y visualizar los resultados, como con la capa de datos para solicitar el almacén o recuperación de datos. Esta capa de datos, o tercer nivel, es donde residen los datos y es la encargada de acceder a los mismos. Se utilizan gestores de base de datos que realicen el almacenamiento y recuperación de información desde la capa de negocio.

Se ve, por último, que la principal ventaja de este tipo de programación es la independencia entre niveles, que permite hacer cambios, en caso de necesidad, sin que esto afecte al resto. Esta abstracción entre las tres capas hace que baste conocer la API existente entre ellas para comunicarlas, y las separa en grupos de trabajo independientes.

4.1. Estructura de la Implementación del Proyecto

Como se ha presentado, se ha procedido al diseño e implementación según la programación por capas.

Para comenzar, en el capítulo 4.2 se muestra cómo se presentan los datos al usuario de forma visual, y cuáles son los elementos gráficos que componen la interfaz de la aplicación de gestión de tareas.

Una vez se conocen las acciones y los aspectos visuales del proyecto, se presenta en el capítulo 4.3 una introducción a las contribuciones más importantes que se han realizado. En la primera de ellas se introduce la importancia de la integración de la aplicación con las redes sociales, y en la segunda se muestra cómo se ha aplicado un control para el nivel de molestia en los avisos de la aplicación. Se presentan además las clases de la librería API de Facebook que permiten después recuperar información de su servidor.

Continuando con la arquitectura de tres niveles, en el capítulo 4.4 se muestra la capa de datos y cómo se ha hecho uso del componente Content Provider. Con este componente es posible guardar y recuperar información de una base de datos de tipo SQLite. Esto es útil para almacenar las tareas de la aplicación de una forma segura y eficaz, a pesar de que se cierre la aplicación o incluso el dispositivo móvil.

Por último en el capítulo 4.5, que corresponde a la capa de negocio, se muestra cómo se utiliza el componente Content Provider para recuperar la información y mostrarla de forma visual en las actividades de la aplicación. Además, se describe cada una de las acciones que podemos realizar en una lista de tareas, y que influyen en la base de datos, como por ejemplo, mostrar, añadir, eliminar y modificar tareas.

Por otro lado se describe la forma en la que se descargan los datos de Facebook. Los eventos que se extraen de Facebook del usuario conectado se adaptan y convierten en tareas y se guardan en la base de datos de la aplicación. Estas tareas que dependen del servidor de esta red social podrán actualizarse de forma manual, o incluso ocultarse si no se desea que se visualicen junto al resto de tareas de la aplicación.

También se presentan las alarmas, y qué proceso se ha seguido para su programación en Android con el componente AlarmManager. Se hace uso además de un componente de terceros llamado WakefullIntentService que permite mantener el sistema despierto y que pueda lanzarse cualquier alarma en cualquier momento. Sin el uso de este componente puede anularse un aviso si el sistema entra en estado de modo de espera.

Se muestra cómo se ha controlado el nivel de molestia, de los avisos de estas alarmas, de un modo contextualizado.

Para finalizar este capítulo muestra la estructura de datos que se ha diseñado para la gestión de las alarmas, activas o canceladas, que cada tarea puede disponer.

4.2. Capa de Presentación

En esta sección se muestra cómo se le presentan los datos de forma visual al usuario. La capa de presentación establece comunicación con la capa de negocio para solicitar el almacén o recuperación de datos según las acciones del usuario.

4.2.1. Visualización de Tareas en la Actividad Principal

Las tareas en la actividad principal se muestran en forma de listado de elementos. Se muestra además un margen superior que contiene un campo de texto editable con un botón anexo para insertar rápidamente una tarea como se ve en el subcapítulo 4.2.2.

Como se aprecia en la Figura 13 cada tarea dispone de los siguientes elementos visuales. Una casilla de verificación que representa el estado de la tarea según esté finalizada o no. Permite la activación o desactivación de la alarma asociada a la tarea, ya que si se finaliza una tarea no será necesario recordarla.

El segundo elemento, si se recorre esta interfaz de izquierda a derecha, es un icono de tipo dinámico que informa de la categoría que esa tarea tiene asociada. Esto es útil para ver rápidamente y de forma gráfica qué tipo de tarea se está mostrando. Por ejemplo, si una tarea pertenece a la categoría de deportes que a su vez tiene asociada una imagen de pelotas de tenis, se mostrará en esta vista esa imagen.

Se muestra además un borde en la imagen con un color relativo a la prioridad que tiene esa tarea. Así de forma indirecta se informa de la prioridad. Los colores disponibles de más prioridad a menos son respectivamente el azul, verde, amarillo, naranja y rojo.

A continuación se muestran dos líneas de texto. La superior es el título de la tarea que tiene asociado el evento de presionado de larga duración para la eliminación de la tarea como se ve en el subcapítulo 4.2.4. La línea de texto inferior muestra la fecha de vencimiento de la tarea, y la hora. La tarea debe completarse antes de esta fecha. Como se ha hecho con la imagen se colorea el texto dependiendo de la prioridad de la tarea.

El último elemento es el icono de una flecha hacia la derecha que permite la edición de la tarea como se puede ver en el subcapítulo 4.2.3.

Figura 13 – Listado de tareas en la actividad principal

4.2.2. Inserción de Tareas

La inserción de tareas se hace de forma directa desde la pantalla principal. El campo de texto editable superior permite escribir el título de la tarea como se observa en la Figura 14. Al pulsar el botón de confirmación se crea una tarea con el resto de parámetros por defecto y se añade de forma inmediata en el listado de tareas.

Figura 14 – Insertar Tareas

4.2.3. Edición de Tareas

Una vez se han insertado previamente tareas en la aplicación es posible editarlas del siguiente modo. Como se observa en la Figura 15 cada tarea que es listada en la pantalla principal dispone de un icono en forma de flecha que al pulsarse hace que se muestre un formulario con los campos editables de cada parámetro de la tarea.

Figura 15 – Edición de una tarea

Es posible editar el título. También la categoría según las que el usuario haya introducido en la aplicación como podemos ver en la siguiente figura. Si no existen se mostrará únicamente la categoría por defecto *Sin Categoría,* donde se incluirá la tarea.

Figura 16 – Edición del campo categoría de una tarea

La prioridad dispone de cinco niveles a elegir: muy baja, baja, media, alta, y muy alta. Los botones donde figuran la fecha y hora del vencimiento de la aplicación, permiten al ser pulsados editar estos campos. Desde cada botón se abre un componente gráfico que permite seleccionar la fecha y hora como se aprecia en la Figura 17.

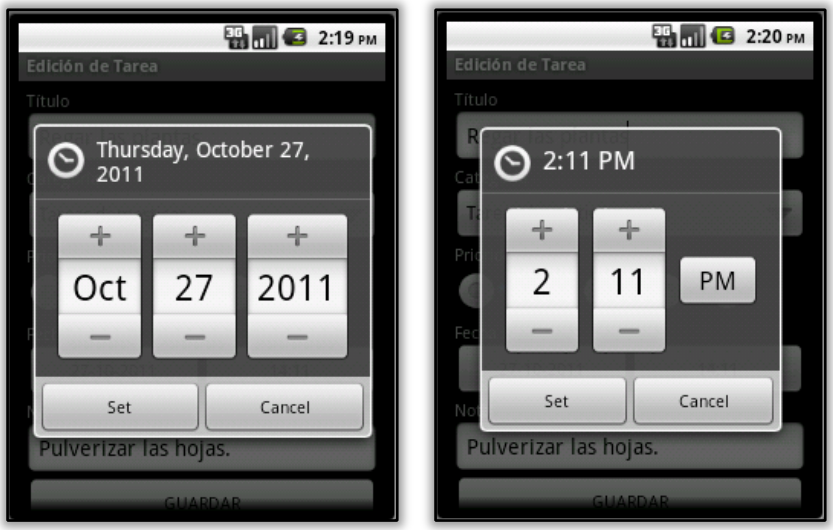

Figura 17 – Selección de fecha y hora de vencimiento de una tarea

Para finalizar la edición es posible añadir una nota con información adicional a la tarea. Los botones Guardar o Cancelar permiten volver a la vista de la pantalla principal guardando los datos en la base de datos, o sin guardar.

4.2.4. Edición de Tareas

Para editar una tarea es necesario pulsar durante varios segundos sobre el título de la tarea que se desea eliminar. Tras esto, aparece un dialogo que requiere confirmar esa eliminación. En caso de responder si, desaparece la tarea del listado de elementos de la pantalla principal como se observa en la siguiente figura.

Figura 18 – Eliminación de una tarea

4.2.5. Menús

Desde cualquier lugar de la aplicación es posible acceder a otras dos secciones desde los iconos del panel desplegable que aparece al pulsar la tecla *menú* del dispositivo móvil. La primera de ellas es la gestión de categorías, y la segunda es la configuración de preferencias.

Además, como se aprecia en la Figura 19, en la pantalla principal aparecen dos iconos más en el menú. Estos permiten cargar u ocultar eventos de Facebook en la lista de elementos de la aplicación si previamente se ha iniciado sesión como usuario de esta red social desde la sección de Preferencias.

Figura 19 – Menú de la pantalla principal

4.2.6. Visualización de Categorías

Desde el menú de la pantalla principal se puede acceder a la sección de categorías. Se muestra aquí un listado de las categorías disponibles. Existen dos categorías por defecto, que son estáticas y no pueden ser editadas. La primera es *Sin Categoría*, que se utiliza cuando el usuario todavía no ha insertado ninguna categoría en la aplicación y sin embargo se insertan tareas. Éstas pertenecen automáticamente a esta categoría por defecto. La segunda categoría es *Facebook*, donde se incluirán las tareas que se importan de los eventos del servidor. Ninguna tarea de usuario puede pertenecer a esta categoría de *Facebook*.

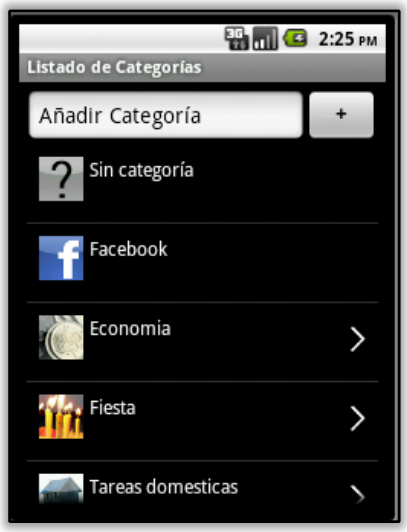

Figura 20 – Visualización de categorías

4.2.7. Edición de Categorías

Del mismo modo que se edita una tarea, como se observa en la Figura 21, es posible modificar el título e imagen asociada a una categoría pulsando el icono en forma de flecha que cada categoría tiene en el listado de categorías.

Figura 21 – Edición de una categoría

4.2.8. Configuración de Preferencias

Desde el menú de la aplicación es posible acceder al panel de preferencias. Como se muestra en la Figura 22 el primer botón superior permite iniciar sesión en Facebook, para posteriormente poder cargar eventos en el listado de tareas.

Al pulsar este botón aparece una ventana de inicio de sesión en Facebook. Tras el rellenado de datos de correo electrónico y contraseña la aplicación valida la nueva sesión abierta y el texto del botón cambia a *logout*. En este estado del botón, al pulsarlo cerrará la sesión que esté abierta en Facebook.

Figura 22 – Configuración de preferencias

Además, se muestran dos casillas de verificación relacionadas con los avisos. La primera de ellas permite deshabilitar todas las alarmas de las tareas, evitando así cualquier tipo de notificación aunque una tarea haya vencido.

La segunda casilla de verificación permite iniciar las alarmas de las tareas existentes en la aplicación al iniciar Android. Esto es útil por ejemplo cuando se reinicia el sistema. Con esta opción nos aseguramos que las alarmas se activan tras un reinicio del sistema sin necesidad de abrir la aplicación de gestión de tareas.

4.2.9. Visualización de Eventos de Facebook en la Actividad Principal

Una vez se ha iniciado sesión en Facebook, es posible hacer uso de los iconos del menú de la pantalla principal *Cargar Facebook* y *Ocultar Facebook*. Así se importan o actualizan los eventos de esta red social que se integran con las demás tareas en la lista de la pantalla principal, como se observa en la Figura 23, o se eliminan de la base de datos de la aplicación según sea necesario.

Un evento de Facebook es de sólo lectura, y no puede ser editado, finalizado, o eliminado, ya que el propietario del evento normalmente es otro usuario que nos ha invitado a él. Solo podemos ocultarlos de nuestro listado de tareas, y así seguirán residiendo en el servidor de Facebook.

Figura 23 – Visualización de tareas junto a eventos de Facebook

4.3. Contribuciones

Las contribuciones de este trabajo tratan de integrar las redes sociales en el dominio de aplicaciones de gestión de tareas de una forma sencilla e intuitiva para el usuario. Por otro lado se introduce la personalización de la aplicación mediante el control del nivel de molestia en los servicios ofrecidos en este tipo de aplicaciones, concretamente en avisos de alarmas de tareas.

4.3.1. Facebook

Las redes sociales se han convertido hoy en día en una de las principales formas de comunicación y de expresión que utiliza la gente en su vida cotidiana, dejando atrás el mensaje de texto SMS, la mensajería instantánea por internet de MSN Messenger, o incluso en ocasiones la propia llamada telefónica.

Con la expansión de los Smartphone, este tipo de comunicación se ha visto asociada directamente al propio dispositivo móvil con acceso a la red. Se convierte en un añadido indispensable en cualquier ámbito tecnológico, y en nuestro caso, en las aplicaciones que tenga instalado el terminal Android para acceder al servicio de la red social.

No es casualidad, pues, que Google valore cada vez más la repercusión social de una página web en los algoritmos de búsqueda de su famoso buscador para el posicionamiento web. Los trabajadores en SEO tienen por ello más en cuenta las redes sociales [24].

Actualmente la red social que más éxito ha cosechado por el número de usuarios es Facebook, que dispone de más de 500 millones.

Hoy en día no existe en Android Market ninguna aplicación gratuita que integre tareas junto a eventos de Facebook. En cuanto a lo novedoso de esta funcionalidad es la posibilidad de integrar en la lista de tareas los eventos de Facebook.

4.3.1.1. Obtención de datos de Facebook

Para la obtención de información de los servidores de Facebook es necesario hacer uso de una serie de librerías que nos permita conectarnos a esta red social y recuperar los eventos del usuario.

Se ha añadido al proyecto una librería que ofrece Facebook bajo licencia Apache, Version 2.0 además de copyright, que contiene herramientas para la gestión del acceso a sus datos, de sesión del usuario, de errores, etc. Según podemos ver en la Figura 24.

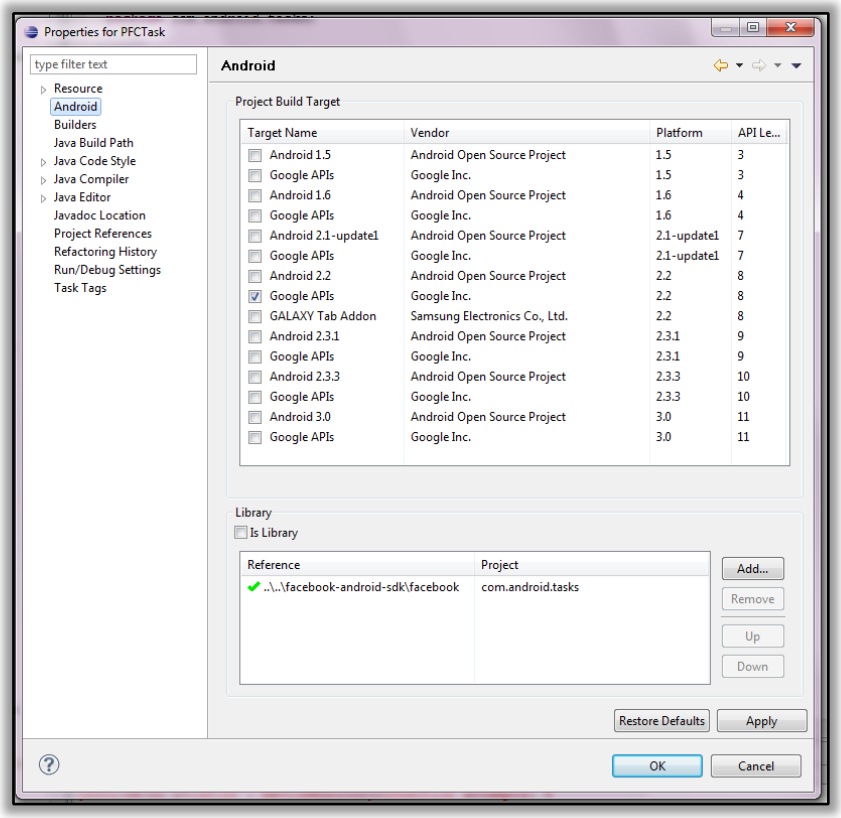

Figura 24- Inserción en nuestro proyecto de Eclipse de la librería SDK de Facebook para Android.

4.3.1.2. Clases de la librería API de Facebook

AsyncFacebookRunner.java

Un ejemplo de implementación de solicitudes de la API asíncrona. Esta clase proporciona la capacidad de ejecutar métodos de la API y obtener la respuesta de la consulta de inmediato, sin bloquear el subproceso de llamada. Esto es necesario para acceder a la API en el hilo de la interfaz de usuario, por ejemplo. La respuesta a la petición se devuelve a la persona que llama a través de una interfaz de devolución de llamada, que el desarrollador debe implementar. Esta implementación de ejemplo, simplemente genera un nuevo hilo para cada solicitud, y hace la llamada a la API de inmediato.

DialogError.java

Encapsulación del diálogo de error.

Facebook.java

Objeto principal de Facebook para interactuar con la API de Facebook para desarrolladores. Proporciona métodos para entrar y salir en el sistema, hacer peticiones utilizando el API REST y de Grafos, e iniciar las interacciones del usuario con la interfaz API (por ejemplo, los pop-ups para la promoción de las credenciales, permisos, etc).

FacebookError.java

Encapsulación de un error de Facebook. Una solicitud que no se haya podido cumplir.

FbDialog.java

Gestiona el dialogo mediante un objeto de tipo WebViewClient para que un usuario acceda al sistema con su cuenta de Facebook.

Util.java

Utilidad de clase de apoyo al objeto de Facebook.

4.3.1.3. Clases de la Aplicación de Gestión de Tareas

Por otro lado en nuestra aplicación se implementan las clases y métodos que hacen posible la conexión con Facebook con las herramientas de la librería anterior.

BaseDialogListener.java y BaseRequestListener.java

Esqueleto de la clase base para objetos de tipo RequestListeners, proporciona el manejo del error por defecto. Las solicitudes deben manejar estas condiciones de error.

LoginButton.java

Inicia el objeto Facebook y actúa como botón de entrada y salida a la sesión de Facebook.

SessionEvents.java

Asocia los eventos recibidos, al objeto de Facebook y actúa en consecuencia, por ejemplo añadiendo o eliminándolos según ocurren.

SessionStore.java

Guarda, restaura, o limpia la sesión en un formato de tipo SharedPreferences con un testigo de acceso, una variable de expiración, y un identificador.

4.3.2. Personalización del Nivel de molestia

Los dispositivos móviles se están convirtiendo cada vez más en la herramienta más importante de comunicación entre los usuarios y los servicios integrados de su entorno. Debido a que este número de servicios se incrementa con el paso del tiempo, se ve necesario un control que personalice estos servicios según las necesidades del usuario y el contexto que lo rodee. Por ejemplo, si el usuario se encuentra en una reunión o en una entrevista de trabajo, y suena una alarma del dispositivo móvil.

Con el fin de evitar esta sobrecarga de información, se presenta un método para personalizar las alarmas según el grado de prioridad de una tarea dada que sea capaz de adaptarse a las necesidades y preferencias del usuario.

Este servicio de alarmas y notificaciones ha sido implementado para reducir al mínimo el esfuerzo e interacción del usuario, y así reducir su nivel de molestia.

Así mismo se ha definido el grado de impertinencia y adaptado el sistema de notificaciones.

4.3.2.1. Detectando las necesidades del usuario

El primer paso en la personalización de los servicios es identificar quién será el usuario, que en nuestro caso se trata de Personas, descritas como gente real que representan un tipo de usuario. Estas Personas describen objetivos del sistema sobre cómo les gustaría utilizar el sistema y qué esperan de éste.

Un ejemplo de usuario puede ser el siguiente. Francisco García es una Persona. Tiene un estilo de vida muy ocupado y a veces olvida realizar tareas importantes que debe hacer como reuniones, o llamadas por teléfono pendientes y otras menos importantes pero esenciales como aniversarios, comprar ciertos elementos, etc. Sus objetivos son la optimización del tiempo, y no olvidar las tareas importantes.

En base al usuario se ha creado, por tanto, un perfil de usuario genérico enfocado a las tareas teniendo en cuenta qué servicios requieren estas Personas para cumplir sus objetivos, y cómo se representan los servicios en términos de intromisión.

El grado de atención que necesita una tarea, la especifica el propio usuario según la prioridad que le asocie como se aprecia en la Figura 25 Edición de la prioridad de una tarea, de esta forma, el usuario es capaz de personalizar el mismo las alarmas.

A pesar de que este criterio depende del usuario individual, se ha diseñado y establecido un marco común para el tipo de usuario Personas según la Tabla 2 – Control de nivel de molestia de las alarmas según la prioridad de una tarea. que se muestra más adelante.

4.3.2.2. Ajuste del nivel de molestia

Para cada interacción que requiera el sistema de notificaciones de tareas se debe asumir el grado de atención que un usuario pueda necesitar.

En el proyecto que se presenta se ha definido esta variable de demanda de atención cuyos extremos son el primer plano y el segundo plano. A una tarea de la que interese obviar su posible vencimiento se le establece la prioridad más baja, por ejemplo si se deben comprar más yogures de los existentes en casa no es importante el aviso, y la notificación se efectuará de forma sigilosa en un segundo plano. El grado de atención en este caso es mínimo.

La interacción que debe haber en el caso opuesto es la más intrusiva, ya que para una tarea urgente es crítico que el usuario sea avisado, mediante una vibración de larga duración, un dialogo que ocupe el primer plano, y un aviso por voz. Siguiendo la taxonomía que se presenta en [21] para la elección de la modalidad o combinación de modalidades que mejor se ajustan a cada situación. Se utiliza así, por tanto, el rango sensitivo del tacto, vista y oído para cumplir el objetivo del servicio de alarmas.

La siguiente Tabla 2 – Control de nivel de molestia de las alarmas según la prioridad de una tarea. especifica el grado de atención que necesita un perfil de usuario según la prioridad de la tarea. Los mecanismos de interacción que se disponen en el dispositivo móvil para este cometido son la vibración, la pantalla y el altavoz.

	Muy Baja	Baja	Media	Alta	Muy Alta
Vibración	No	Vibración sencilla	Vibración de patrón discontinuo	Vibración continua de 5 segundos	Vibración continua de 10 segundos
Visual	Barra de Notificación	Barra de Notificación	Barra de Notificación	Dialogo	Dialogo
Audio	No	No	No	Sonido corto “Beep”	Lectura de la tarea por voz.

Tabla 2 – Control de nivel de molestia de las alarmas según la prioridad de una tarea.

Podemos ver en la Figura 25 Edición de la prioridad de una tarea cómo puede el usuario llevar a cabo la personalización en el sistema que se ha desarrollado con la edición del campo Prioridad.

Figura 25 Edición de la prioridad de una tarea

4.4. Capa de datos

Este nivel de abstracción de la capa de datos permite el almacenamiento de datos y el acceso a los mismos. Si la capa de negocio requiere guardar o consultar información debe hacer solicitudes a esta capa de datos, que es finalmente la encargada de gestionar la base de datos.

Como ya se ha visto anteriormente uno de los componentes principales de Android es el Content Provider. Gracias a esto podemos almacenar la información en un solo archivo de sistema como base de datos SQLite pudiendo acceder a ésta mediante los métodos propios de ContentResolver. Un ContentProvider permite que este acceso a una misma base de datos puedan hacerlo distintas aplicaciones.

SQLite [40] es un sistema de gestión de base de datos relacional, que a diferencia de los sistemas de gestión de bases de datos cliente-servidor tiene una latencia más reducida en el acceso a la base de datos y el conjunto de la base de datos es guardado como un solo fichero estándar en la máquina host.

El diseño de la base de datos dispondrá de dos tablas con los campos que se muestran en la Tabla 3.

Tasks	
_ID	INTEGER. Clave primaria autoincremental que identifica una tarea.
TITULO	VARCHAR. Título de la tarea.
CATEGORIA	INTEGER. ID referencia correspondiente a la categoría de la tabla "categories"
FINALIZADA	INT. Se considera finalizada con el valor 1, y no finalizada con 0.
PRIORIDAD	INT. Prioridades: 5: Muy alta; 4: Alta; 3: Media; 2:Baja; 1:Muy Baja.
VENCIMIENTO	TEXT. Fecha de vencimiento de la tarea según el estándar ISO8601: yyyy-MM-dd'T'HH:mm:ss:SSS
NOTAS	TEXT. Cualquier nota que el usuario desee guardar
LATITUD	REAL.
LONGITUD	REAL.

→

Categories	
_ID	INTEGER. Clave primaria autoincremental que identifica una categoría.
TITULO	VARCHAR. Título de la categoría.
IMAGEN	INTEGER. ID correspondiente a la imagen que se asocie con cada categoría. Se gestiona mediante software con las imágenes contenidas en la carpeta /res/
DESCRIPCION	VARCHAR. Descripción asociada a la categoría.

Tabla 3 –Tablas de tareas y categorías de la base de datos de la aplicación.

4.4.1. SQLiteOpenHelper

Para la creación de la base de datos y el control de la versión de ésta, se ha creado la clase TasksData que extiende a SQLiteOpenHelper. Esta clase ayudará a conectar con la base de datos, y en caso de que no exista puede crear una nueva. Hace fácil su uso al objeto de tipo Content Provider.

```
public class TasksData extends SQLiteOpenHelper {
   private static final String DATABASE_NAME = "tasks.db";
   private static final int DATABASE_VERSION = 2;

   public TasksData(Context ctx) {
      super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
   }
   @Override
   public void onCreate(SQLiteDatabase db) {
      db.execSQL("CREATE TABLE " + TABLE_NAME_TASKS
               + " (" + _ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
               + TITULO + " VARCHAR, "
               + CATEGORIA + " INTEGER, "
               + FINALIZADA + " INT(1), "
               + PRIORIDAD + " INT(1), "
               + VENCIMIENTO + " TEXT, "
               + NOTAS + " TEXT, "
               + LATITUD + " REAL, "
               + LONGITUD + " REAL);");
      db.execSQL("CREATE TABLE " + TABLE_NAME_CATEGORIES
               + " (" + _ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
               + TITULO_CAT + " VARCHAR, "
               + IMAGEN + " INT(2), "
               + DESCRIPCION + " VARCHAR);");
   }
…
```

TasksData.java

En el constructor de TasksData se hace una llamada al constructor de la clase padre, bien para crear la base de datos llamada “tasks.db” cuya versión sea 2, o bien para conectar con ella en un momento posterior.

Es posible modificar la estructura de la base de datos con el método onUpgrade de esta misma clase para añadir campos, o crear nuevas tablas.

4.4.2. ContentProvider

Un ContentProvider es un componente que se ejecuta en un segundo plano que permite gestionar la base de datos y su acceso entre distintas aplicaciones. Su estructura ha sido diseñada para realizar las operaciones de crear, leer, actualizar y eliminar.

En el modelo de seguridad de Android los archivos de una aplicación no pueden ser leídos ni escritos por cualquier otra aplicación, ya que disponen de su propio ID de usuario Linux y directorio de datos, y su propio espacio de memoria reservado. Aunque las aplicaciones de Android pueden comunicarse con otras mediante dos métodos.

- Inter-Process Communication (IPC): Un proceso declara un API arbitrario utilizando el Android Interface Definition Language (AIDL), y la interfaz iBinder. Los parámetros son enviados de forma segura y eficiente entre procesos cuando la API es requerida.
- Content Provider: Los procesos se registran así mismos en el sistema como proveedores de ciertos tipos de datos. Cuando se quieren leer los datos son llamados mediante una API fijada de Android.

La clase TasksProvider extiende a ContentProvider.

Uri

Las direccioines URI identifican cualquier recurso en la base de datos, u otro posible contenido, para que puedan ser accedidos.

Por ejemplo, si se quiere acceder a la tarea cuya ID es la número 5, se deberá hacer mediante la URI: “content://com.android.tasks/tasks/5”. Esta dirección se descompone en los siguientes elementos:

- Prefijo del protocolo “content://”
- Autoridad “com.android.tasks”. Se utiliza además en el archivo Android Manifest.
- Recurso “tasks”. Especifica la tabla a la que queremos acceder.
- Identificador “5”. Este identificador concreta un recurso individual dentro de la colección “tasks”, en este caso el número 5. Si no se especificara se accedería a todo el contenido de “tasks”.

UriMatcher

Para poder analizar la dirección URI de los recursos necesarios se crea una variable estática para un UriMatcher, ya que no cambiará a lo largo de la vida de la aplicación.

```
public class TasksProvider extends ContentProvider {
   private static final int TASKS = 1;
   private static final int TASKS_ID = 2;
   private static final int CATEGORIES = 3;
   private static final int CATEGORIES_ID = 4;
   private static final UriMatcher sURIMatcher = new UriMatcher(UriMatcher.NO_MATCH);
…
public boolean onCreate() {
      uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
      uriMatcher.addURI(AUTHORITY, "tasks", TASKS);
      uriMatcher.addURI(AUTHORITY, "tasks/#", TASKS_ID);
      uriMatcher.addURI(AUTHORITY, "categories", CATEGORIES);
      uriMatcher.addURI(AUTHORITY, "categories/#", CATEGORIES_ID);
      tasks = new TasksData(getContext());
      return true;
   }
…
```

TasksProvider.java

Un objeto de la clase UriMatcher es de utilidad para comparar direcciones URI en los ContentProviders. En la creación de este UriMatcher se añaden las URIs que más adelante podrán ser utilizadas por el ContentProvider.

Por ejemplo “uriMatcher.addURI("com.android.tasks", "tasks", 1);” identifica todos los registros del recurso tasks, y “uriMatcher.addURI("com.android.tasks", "tasks/#", 2);“ solamente identifica un registro en concreto. La última variable es un código que debe devolver UriMatcher.match(Uri uri) cuando identifique esta dirección URI.

Método getType

Nuestro ContentProvider utiliza los MIME types que se han establecido como un recurso único “vnd.android.cursor.dir/task”, o como una colección de recursos “vnd.android.cursor.item/task”. Este método getType es útil para distinguir el tipo de la dirección Uri que llega al ContentProvider.

```
…
@Override
   public String getType(Uri url){
       int match = sURIMatcher.match(url);
       switch (match) {
           case TASKS:
               return CONTENT_TYPE_TASKS;
           case TASKS_ID:
               return CONTENT_ITEM_TYPE_TASKS;
           case CATEGORIES:
               return CONTENT_TYPE_CATEGORIES;
           case CATEGORIES_ID:
               return CONTENT_ITEM_TYPE_CATEGORIES;
           default:
               throw new IllegalArgumentException("Unknown URI " + url);
       }
   }
…
```

TasksProvider.java

Método leer

Para acceder a la base de datos es necesario concretar qué tipo de recurso se desea consultar, y de qué forma se requieren los datos. Esto se hace mediante una llamada al método Query del objeto ContentProvider.

Los parámetros del método Query son:

- URI. Es una dirección que identifica el recurso que se quiere leer utilizando el esquema content://
- Proyección. Una colección de cadenas con las columnas o campos de la tabla que se quieran leer. Con un valor *null*, devuelve todas las columnas.
- Selección. Es un filtro que especifica qué filas queremos leer. Tiene el formato de una consulta WHERE de SQL. Si este parámetro es null se devuelven todas las filas.
- Argumentos de Selección. Se pueden incluir en la selección, los cuales serán reemplazados por los valores de los argumentos de selección.
- Orden. Se especifica cómo ordenar las filas. Formateado como una clausula SQL ORDER BY. Con el valor null se ordena por defecto, lo que puede ser de forma desordenada.

```
…
   @Override
   public Cursor query(
                   Uri uri,
                   String[] projection,
                   String selection,
                   String[] selectionArgs,
                   String orderBy) {

      SQLiteDatabase db = tasks.getReadableDatabase();
      Cursor cursor = null;
      int matchu = uriMatcher.match(uri);
      if (matchu == TASKS_ID || matchu == TASKS) {
          if(matchu == TASKS_ID){
                 long id = Long.parseLong(uri.getPathSegments().get(1));
                 selection = appendRowId(selection, id);
          }
         cursor = db.query(TABLE_NAME_TASKS, projection, selection, selectionArgs,
null, null, orderBy);
          cursor.setNotificationUri(getContext().getContentResolver(), uri);
          return cursor;
      }
      else if (matchu == CATEGORIES_ID || matchu == CATEGORIES) {
          if(matchu == CATEGORIES_ID){
               long id2 = Long.parseLong(uri.getPathSegments().get(1));
               selection = appendRowId(selection, id2);
          }
           cursor = db.query(TABLE_NAME_CATEGORIES, projection, selection,
selectionArgs, null, null, orderBy);
           cursor.setNotificationUri(getContext().getContentResolver(), uri);
           return cursor;
       }
       return null;
   }
…
```

TasksProvider.java

En el fragmento de código anterior se observa cómo se devuelve un objeto de tipo Cursor con los datos de la consulta realizada a la base de datos.

Un Cursor es un objeto cuya interfaz proporciona un acceso aleatorio de lectura y escritura para el conjunto de resultados devueltos por una consulta de base de datos.

Método insertar

Para poder insertar datos en la base de datos es necesario especificar la dirección URI del recurso donde se desean incluir, y su conjunto de valores que se recogen en el objeto de tipo ContentValues. Mediante el método *put* se agregan valores a ContentValues según el parámetro par "clave, valor".

Este método devuelve la dirección URI del elemento que se acaba de insertar, ya que hasta que no se realiza esta acción es desconocida. Sus parámetros son:

- Identificador URI de la tabla a insertar el elemento.
- Los valores iniciales para la nueva fila insertada guardados en un objeto de tipo ContentValues.

```
…
@Override
   public Uri insert(Uri uri, ContentValues values) {
      SQLiteDatabase db = tasks.getWritableDatabase();
      Uri newUri;

      int match = uriMatcher.match(uri);
      // Validate the requested uri
      if (match != TASKS) {
          if (match != CATEGORIES) {
                throw new IllegalArgumentException("Unknown URI " + uri);
          }
      }
      if (match == TASKS) {
            // Insert into database
            long id = db.insertOrThrow(TABLE_NAME_TASKS, null, values);
            // Notify any watchers of the change
            newUri = ContentUris.withAppendedId(CONTENT_URI_TASKS, id);
      }else{
          long id2 = db.insertOrThrow(TABLE_NAME_CATEGORIES, null, values);
          // Notify any watchers of the change
          newUri = ContentUris.withAppendedId(CONTENT_URI_CATEGORIES, id2);
      }
      getContext().getContentResolver().notifyChange(newUri, null);
      return newUri;
   }
…
```

TasksProvider.java

Método actualizar

Otra de las posibles acciones a realizar es la actualización de datos. Además de la dirección URI y el conjunto de valores nuevos guardados en un objeto de tipo ContentValues, ya vistos en el método insertar, para la modificación de recursos es necesaria una cadena que actúe como filtro para seleccionar las filas a actualizar.

Este método devuelve el número de registros afectados por la actualización. Recibe los siguientes parámetros:

- La URI a modificar.

- Los valores de los nuevos campos.
- Un filtro a aplicar a las filas antes de la actualización. Tiene formato de la cláusula SQL WHERE.

```
…
@Override
   public int update(
                   Uri uri,
                   ContentValues values,
                   String selection,
                   String[] selectionArgs) {

      SQLiteDatabase db = tasks.getWritableDatabase();
      int count;
      switch (uriMatcher.match(uri)) {
      case TASKS:
         count = db.update(TABLE_NAME_TASKS, values, selection, selectionArgs);
         break;
      case TASKS_ID:
         long id = Long.parseLong(uri.getPathSegments().get(1));
         count = db.update(TABLE_NAME_TASKS, values, appendRowId(selection, id),
selectionArgs);
         break;
      case CATEGORIES:
          count = db.update(TABLE_NAME_CATEGORIES, values, selection, selectionArgs);
          break;
       case CATEGORIES_ID:
          long id2 = Long.parseLong(uri.getPathSegments().get(1));
          count = db.update(TABLE_NAME_CATEGORIES, values, appendRowId(selection,
id2), selectionArgs);
          break;
      default:
         throw new IllegalArgumentException("Unknown URI " + uri);
      }

      // Notify any watchers of the change
      getContext().getContentResolver().notifyChange(uri, null);
      return count;
   }
…
```

TasksProvider.java

Método eliminar

El último método para gestionar los datos del objeto de tipo ContentProvider es el de eliminar. Su funcionamiento es similar al resto. Se identifica el recurso mediante la URI y se actúa eliminándolo. Sus parámetros son:

- URL de la fila a eliminar
- Un filtro a aplicar a filas antes de la eliminación. Su formato es como una clausula SQL WHERE.

```
…
@Override
   public int delete(
                   Uri uri,
                   String selection,
                   String[] selectionArgs) {

      SQLiteDatabase db = tasks.getWritableDatabase();
      int count;
      switch (uriMatcher.match(uri)) {
      case TASKS:
         count = db.delete(TABLE_NAME_TASKS, selection, selectionArgs);
         break;
      case TASKS_ID:
         long id = Long.parseLong(uri.getPathSegments().get(1));
         count = db.delete(TABLE_NAME_TASKS, appendRowId(selection, id),
```

```
selectionArgs);
            break;
        case CATEGORIES:
            count = db.delete(TABLE_NAME_CATEGORIES, selection, selectionArgs);
            break;
        case CATEGORIES_ID:
            long id2 = Long.parseLong(uri.getPathSegments().get(1));
            count = db.delete(TABLE_NAME_CATEGORIES, appendRowId(selection, id2),
selectionArgs);
            break;
        default:
           throw new IllegalArgumentException("Unknown URI " + uri);
        }

        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
…
```

TasksProvider.java

Declaración en AndroidManifest.xml

Finalmente, por cuestiones de seguridad, se debe añadir el proveedor de contenidos como componente dentro del fichero de Manifiesto AndroidManifest.xml añadiendo el elemento <provider> como hijo del elemento <aplication> con los siguientes atributos:

- android:name. Debe mostrar el nombre de la clase del ContentProvider.
- android:authority. Como medida de seguridad se permite la utilización del ContentProvider únicamente dentro del páquete que especifica este atributo.

```
…
<application android:icon="@drawable/icon" android:label="@string/app_name">
      <provider android:name=".TasksProvider" android:authorities="com.android.tasks"
/>
…
```

AndroidManifest.xml

4.5. Capa de Negocio

En este nivel de programación se establece una serie de funciones y reglas que permitan acceder a los datos de la base de datos de la capa de datos, y a su vez mostrar y recuperar información del usuario presentando estos datos en la capa de presentación. En este capítulo se muestra cómo se han implementado estas funciones.

4.5.1. Utilización de ContentProvider desde una Actividad

El objeto de tipo ContentProvider de una aplicación tiene su propio ciclo de vida, no es necesario inicializarlo, por ejemplo, cada vez que se crea una Actividad. Accederemos a él mediante un objeto ContentResolver único, también gestionado por Android.

4.5.1.1. Añadir tarea

Para la entrada de tareas se muestra en la actividad principal un campo de texto junto a un botón. Con este botón se puede insertar directamente una tarea con el título que sea escrito, completándose los demás parámetros por defecto. El código de dicho botón se muestra a continuación.

```
…
            …
           // Botón añadir nueva tarea
           Button b = (Button)findViewById(R.id.Button01);
           b.setOnClickListener(new View.OnClickListener() {
                  public void onClick(View v){
                         EditText et=(EditText)findViewById(R.id.EditTextADDTASK);
                         addTask(et.getText().toString());
                         et.setText("Añadir Tarea");
                  }
           });
…
```

Tasks.java

Figura 26 - Campo de texto editable y botón para insertar tarea

El evento *onClick* del botón conduce la ejecución al método *addTask* que completa de forma automática el resto de campos de la tarea por defecto mediante una colección de datos en un objeto de tipo ContentValue y los añade en forma de registro al objeto ContentProvider. Como se ha explicado anteriormente para acceder a un ContentProvider se necesitan utilizar los métodos de un ContentResolver.

```
        …
        ContentValues values = new ContentValues();
        values.put(TITULO, titulo);
        values.put(CATEGORIA, 1);
        values.put(FINALIZADA, 0);
        values.put(PRIORIDAD, 3);
        String fechaNow = getNow();
        values.put(VENCIMIENTO, fechaNow);
        values.put(NOTAS, "Sin nota");
        values.put(LATITUD, 1.00);
        values.put(LONGITUD, 1.00);
        …
        // Añado la tarea
        Uri urivar = getContentResolver().insert(CONTENT_URI_TASKS, values);
        …
```

Tasks.java

La categoría identificada con el valor de ID 1 está por defecto en la aplicación con el fin de clasificar las tareas *Sin categoría*. El valor 0 del campo llamado *finalizada* indica que no lo está. Se incluye una prioridad media con el valor 3, y la fecha actual con el mes incrementado en uno. Es decir, se da por defecto un mes de plazo para cada tarea nueva, suficiente para ser modificado en caso necesario. Esta fecha se obtiene en formato ISO8601 que se gestiona mediante cadenas en la base de datos. Se observa que la variable *fechaNow* es de tipo String.

4.5.1.2. Eliminar tarea

Cada tarea puede ser eliminada presionando durante varios segundos su título. Esta cadena tiene asociada una llamada al dialogo de confirmación para eliminar la tarea.

```
…
tv.setOnLongClickListener(new OnLongClickListener() {
        public boolean onLongClick(View v) {
                TextView vv = (TextView)v;
                String titulo = (String) vv.getText();
                int id = (Integer) v.getTag();
                Tasks.dialogoEliminarTarea(id, titulo);
                return true;    }    });
…
```

Tasks.java

Tras confirmar el dialogo de la Figura 27, se llama a la función *deleteTask* que recupera el ContentResolver y elimina la tarea del siguiente modo:

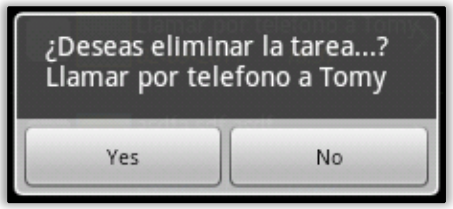

Figura 27 - Dialogo de confirmación de eliminación de tarea

```
private void deleteTask(int id){
        getContentResolver().delete(CONTENT_URI_TASKS, "_ID="+id, null);
}
```

Tasks.java

Los parámetros que necesita este método *deleteTask* son la URI de las tareas, distinta a la de las categorías. Esta constante es content://com.android.tasks.tasks. Es importante, además, enviar el parámetro ID único de la tarea que queremos eliminar, de lo contrario se eliminarían todos los registros de la colección tasks.

4.5.1.3. Actualizar tarea

Una tarea puede ser editada desde la actividad principal mediante un icono de flecha que cada una dispone. El evento onClick asociado lleva a otra actividad del siguiente modo:

1. Actividad A. Se recupera el ID propio de la tarea que ha recibido el clic para identificarla en la base de datos.
2. Actividad A. A partir del ID se recuperan todos los campos de esa tarea. Eso es llamando al método *getTask* que devuelve un objeto de tipo Cursor con una lista de las tareas que coinciden con esa id.
3. Actividad A. Se crea un objeto de tipo Intent cuyo destino será la actividad de la clase *TaskEditing*, y se le añade cada valor de la tarea con el método *putExtra*.
4. Actividad A. Se pasa a otra actividad, que consta de un formulario donde poder editar cada uno de los datos.

Figura 28 - Formulario de edición de una tarea

Pueden verse estos pasos reflejados en el siguiente código:

```
…
b.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
                int ide = (Integer) v.getTag();
                Cursor c = getTask(ide);
                c.moveToFirst();
                String titu = c.getString(c.getColumnIndexOrThrow(TITULO));
                int cate = c.getInt(c.getColumnIndexOrThrow(CATEGORIA));
                int prio = c.getInt(c.getColumnIndexOrThrow(PRIORIDAD));
                String nota = c.getString(c.getColumnIndexOrThrow(NOTAS));
                String dateANDtime = c.getString(c.getColumnIndexOrThrow(VENCIMIENTO));
                String dateSPLITtime[] = splitDate(dateANDtime);
                Intent myIntent = new Intent(myContext, TasksEditing.class);
                myIntent.putExtra("id", ide);
                myIntent.putExtra("titulo", titu);
                myIntent.putExtra("categoria", cate);
                myIntent.putExtra("prioridad",prio);
                myIntent.putExtra("vencimiento", dateSPLITtime);
                myIntent.putExtra("notas", nota);
                //Lanzo la nueva actividad TasksEditing
                startActivityForResult(myIntent,TASKS_REQUEST);
        }
  });
…
```

Tasks.java

5. Actividad B. Una vez se ha cambiado a la actividad de edición de la tarea se recupera el objeto Intent que se ha enviado desde la actividad principal A con el método *getIntent*. Los valores que anteriormente se han añadido se recuperan con métodos como *getIntExtra* o *getStringExtra*, dependiendo del tipo del parámetro.
6. Actividad B. Al hacer clic en el botón *Guardar* se asegura que la fecha de vencimiento no sea anterior a la actual, y que el título de la tarea no es vacío. Se reconstruye el intent con los nuevos valores de cada campo del mismo modo que antes, con el método putExtra, y se vuelve a la actividad principal con estas líneas:

```
        setResult(RESULT_OK,obtained);
        finish();
```

TasksEditing.java

7. Actividad A. Al volver a la actividad principal se entra de forma automática al método *onActivityResult*. Esto es porque la intención de ir a la actividad de edición de tareas es la de obtener unos resultados.
8. Actividad A. Tras obtener los datos del objeto de tipo Intent se llama al método *updateTask* que gestiona la actualización de la tarea aportando los valores nuevos en un ContentValue al ContentResolver como sigue:

```
private int updateTask(int id, ContentValues values){
return getContentResolver().update(CONTENT_URI_TASKS, values, "_ID="+id, null);
}
```

TasksProvider.java

4.5.1.4. Mostrar tareas

El enlace a datos permite conectar una actividad con la base de datos en pocas líneas de código. Para ello la clase *Tasks* implementa a ListActivity. Mediante el método showTasks(cursor) se puede mostrar una lista de tareas extraídas del ContentProvider.

```
private void showTasks(Cursor cursor) {
 // Set up data binding
SimpleCursorAdapter adapter =
              new SimpleCursorAdapter(this, R.layout.item, cursor, FROM, TO);
…
setListAdapter(adapter);
}
```

Se crea un SimpleCursorAdapter para el cursor, que contiene el listado de tareas de la base de datos. Este adaptador actúa como enlace a tiempo real conectando la vista con su fuente de datos. Sus cinco parámetros son los siguientes:

- Context. Una referencia a la actividad actual.
- Layout. Un recurso que define la vista para un único elemento de la lista
- Cursor. El conjunto de datos cursor.

- From. La lista del nombre de las columnas de donde los datos provienen.
- To. La lista de las vistas donde los datos van.

Se ha definido la disposición de un elemento de la lista en el archivo /res/layout/ítem.xml. El parámetro array “TO” anterior coincide con el ID de la fila del layout.

```
<?xml version="1.0" encoding="utf-8"?>
       <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
                       android:layout_width="fill_parent"
                       android:layout_height="fill_parent"
                       android:orientation="horizontal"
                       android:padding="10sp">

                   <CheckBox android:id="@+id/finalizada"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:checked = "false" />

                   <ImageView android:id="@+id/categoria"
                       android:layout_width="38sp"
                        android:layout_height="38sp"
                        android:padding="1sp"
                        android:layout_marginTop="3sp"
                       android:layout_toRightOf="@+id/finalizada"
                       android:src='@drawable/categoria01sincategoria'>
                       </ImageView>
                   <TextView android:id="@+id/rowid"
                       android:layout_width="wrap_content"
                       android:layout_height="wrap_content"
                       android:layout_toRightOf="@+id/categoria"/>
                   <TextView android:id="@+id/titulo"
                           android:layout_width="wrap_content"
                           android:layout_height="wrap_content"
                           android:layout_toRightOf="@+id/rowid"
                           android:textColor="#FFF"
                           android:textSize="16sp"
                           android:paddingLeft="3sp"
                           android:maxLines="2" />

                   <TextView android:id="@+id/vencimiento"
                           android:layout_width="fill_parent"
                           android:layout_height="wrap_content"
                           android:paddingLeft="2sp"
                           android:ellipsize="end"
                           android:singleLine="true"
                           android:textStyle="italic"
                           android:layout_below="@+id/titulo"
                           android:layout_toRightOf="@+id/categoria"/>

                      <Button
                               android:id="@+id/buttonrightarrow"
                               android:layout_width="wrap_content"
                               android:layout_height="wrap_content"
                               android:background="@drawable/rightarrow"
                            android:layout_alignParentRight="true"/>
</RelativeLayout>
```

/res/layout/ítem.xml

Por otro lado, en la capa visual de la actividad principal que se quiere listar elementos se ha incluido el siguiente código.

```
            <ListView android:id="@android:id/list"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"/>
            <TextView android:id="@android:id/empty"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/empty"/>
```

TasksProvider.java

Puesto que la actividad implementa la clase ListActivity, Android gestiona que se muestre la vista android:id/list si dispone de elementos, y en caso contrario que se muestre android:id/empty que contiene la cadena del recurso @string/empty que indica que no existen todavía datos para mostrar.

```
<resources>
    <string name="app_name">Listado de Tareas</string>
    <string name="empty">No hay tareas para mostrar.</string>
…
```

/res/values/strings.xml

Adaptación de los datos a mostrar

Puesto que los datos obtenidos para cada tarea deben ser interpretados, se ha especificado con el método *setViewBinder* cómo debe hacerse.

```
…
adapter.setViewBinder(new SimpleCursorAdapter.ViewBinder() {
    public boolean setViewValue(View view, Cursor cursor, int columnIndex) {
…
```

Tasks.java

Android gestiona de forma automática el recorrido del cursor siguiendo un orden de filas y de columnas. Para un registro concreto por ejemplo se llama de forma iterativa la función *setViewValue* en cada columna, y en función a la columna que se esté tratando se asignan los valores adecuados a cada vista, del elemento de la lista, tal como sigue.

1. Prioridad. Se guarda el valor en una variable estática para utilizarse en las siguientes columnas.
2. Categoría. Se muestra de forma dinámica el recurso de la imagen que la categoría tiene asociada. Según la prioridad guardada se muestra un borde de color rojo, naranja, amarillo, verde o azul.
3. _ID y Flecha. Se guarda el valor en una variable estática del ID de la tarea para su posterior uso. La flecha tiene el evento *onClick* para poder editar la tarea del ID dado. Como se ha explicado en el punto *Actualizar tarea* se genera un objeto de tipo Intent con los campos de la tarea y se cambia a la actividad que edita la tarea. La flecha no se muestra si la tarea corresponde a un evento de Facebook, estas tareas no son editables.
4. Título. Se le asocia al título la cadena correspondiente. Se añade un evento *onLongClick* para que presionando durante una larga se muestre un dialogo de eliminación de la tarea. Se hace uso para su identificación del _ID guardado en

el punto 3. Si la tarea corresponde a un evento de Facebook no se permite su eliminación.

5. Finalizada. Una casilla de verificación permite modificar el campo *finalizada* gracias al _ID del punto 3.
6. Vencimiento. Se muestra la fecha y la hora del vencimiento de la tarea en formato dd/mm/aaaa hh:mm. Dependiendo de la prioridad guardada en el punto 1 se colorea el texto en rojo, naranja, amarillo, verde o azul.

Tras todos estos pasos un elemento de la lista queda dispuesto según la siguiente figura:

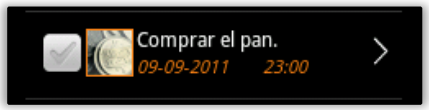

Figura 29 – Disposición gráfica de una tarea como elemento en la lista de tareas

4.5.2. Descarga de Facebook en el ContentProvider local

La conexión entre la aplicación y los eventos de Facebook se realiza de forma activa por parte del usuario. Primero desde la actividad de preferencias, es posible entrar en Facebook con una cuenta personal haciendo clic en el botón de inicio de sesión. Tras esto la sesión permanecerá abierta mientras Facebook lo permita mediante el campo de expiración del objeto Facebook de su clase homónima.

Tras esto es posible mostrar u ocultar los elementos de la lista que sean eventos de Facebook mediante los botones del menú *Cargar Facebook* y *Ocultar Facebook* respectivamente.

4.5.2.1 Cargar Facebook

En la clase *Preferences* se mantienen por tanto todos los objetos referentes a Facebook asociados al botón de inicio de sesión, o login. Cuando se pulsa en el menú el elemento *Cargar Facebook* desde la actividad principal, se accede a la configuración de preferencias y se comprueba si se ha iniciado previamente la sesión, si esto no es así se avisa al usuario.

Si la sesión es correcta desde la actividad principal Tasks se crea un objeto de tipo AsyncFacebookRunner, desde donde es posible hacer la consulta al servidor Facebook para obtener los eventos.

```
mAsyncRunner = new AsyncFacebookRunner(Preferences.mFacebook);
mAsyncRunner.request("me/events", new SampleRequestListener());
```

Tasks.java

Este objeto ejecuta un hilo en segundo plano, ya que la respuesta del servidor no es inmediata. Así la aplicación puede seguir funcionando y no quedarse bloqueada.

Una vez el servidor de Facebook ha resuelto la consulta de eventos mediante la cadena *me/events* devuelve un objeto de tipo JSON que se debe desglosar. JSON es un formato ligero para el intercambio de datos. Se muestra un ejemplo de evento real en este formato:

```
{
  data: [
   {
      name: "Referéndum 15 de octubre",
      start_time: "2011-10-15T00:00:00",
      end_time: "2011-10-15T23:30:00",
      location: "Desde tu casa con DNI-e o voto tradicional en puestos de votación
físicos que se habilitarán para facilitar la participación",
      id: "138185479589316",
      rsvp_status: "attending",
   }
  ]
}
```

Ejemplo de datos en formato JSON

En la clase *SampleRequestListener* se ha implementado el método *onComplete*, que se ejecuta justo cuando termina la petición a Facebook, desde la que se extraen los datos de los campos *name*, *end_time* y *location*.

```
public class SampleRequestListener extends BaseRequestListener {
       public void onComplete(final String response, final Object state) {
           try {
               // process the response here: executed in background thread
               JSONObject json2 = Util.parseJson(response);
               JSONArray d = json2.getJSONArray("data");
               int l = (d != null ? d.length() : 0);

                  eventF = new String [l][3];
               for (int i=0; i<l; i++) {
                   JSONObject o = d.getJSONObject(i);
                   // titulo
                   if(o.has("name")){
                                 eventF[i][0] = o.getString("name");
                   }else{
                         eventF[i][0] = "Sin título";
                   }
…
```

Tasks.java

Se añade a la base de datos una nueva tarea por cada fila del objeto JSON con los atributos *título*, *fecha de vencimiento* y *notas* respectivamente. Esto se hace recorriendo el conjunto de datos que se ha creado en el método *onComplete* de la clase *SampleRequestListener* y llamando de forma iterativa al método insertar del objeto ContentResolver tan sólo si el evento no es anterior a la fecha actual.

```
getContentResolver().insert(CONTENT_URI_TASKS, values);
```

Tasks.java

4.5.2.2 Ocultar Facebook

Si no se desea seguir viendo los eventos de Facebook intercalados entre las tareas se debe hacer clic en el elemento del menú *OcultarFacebook* de la actividad principal de la aplicación. Se muestra al instante que desaparecen todos los elementos de la lista que pertenecen a Facebook. Esto no termina la sesión que el usuario haya iniciado en Facebook, simplemente elimina la copia local de los elementos que anteriormente se han descargado.

Para ello se eliminan de la base de datos todas las tareas que pertenezcan a la categoría cuyo ID es 2, que por defecto es *Facebook*. Una tarea de usuario no podrá pertenecer a esta categoría, ni es posible tampoco editarla ni eliminarla.

```
protected void deleteTasksOfCategory(String id){
    getContentResolver().delete(CONTENT_URI_TASKS, "CATEGORIA="+id, null);
}
```

Tasks.java

El segundo parámetro *CATEGORIA=2* actúa como una cláusula WHERE en SQL. Al indicar el identificador de recurso URI “content://com.android.tasks/tasks/”, se accede a toda la colección del recurso, y la operación se ejecuta sobre toda la colección y no sobre una tarea individual.

4.5.3. Alarmas

Uno de los principales problemas para la creación de alarmas es la gestión de energía. Android es un sistema diseñado específicamente para baterías de corta duración, y por ello es necesario que las alarmas actúen aunque el dispositivo se encuentre en modo de espera. Este funcionamiento es el mismo, por ejemplo, que una aplicación de correo. Se desea seguir recibiendo emails a pesar de que el dispositivo esté en modo de espera.

Se ha utilizado para ello el servicio WakefulIntentService de CommonsWare Android Components. Este componente openSource mejora el funcionamiento de la clase IntentService de Android que mantiene el dispositivo despierto mientras se ejecuta la tarea en un segundo plano. Resulta útil en este caso para programar alarmas a través de la clase AlarmManager.

4.5.3.1 Registro de alarmas

Para la activación de alarmas se hace uso de la clase AlarmManager, que necesita los siguientes componentes:

- PendingIntent. Es una descripción de un Intent y la acción objetivo a realizar con él. Se instancia con los métodos getActivity, getBroadcast, y getService. Nuestro intent apunta a la clase OnAlarmReceiver.
- La fecha y hora a la que debe realizar la acción la alarma. En formato Calendar.

```
AlarmManager mgr=(AlarmManager)myContext.getSystemService(Context.ALARM_SERVICE);
            Intent i=new Intent(myContext, OnAlarmReceiver.class);
            i.putExtra("id", id);
            i.putExtra("titulo",titulo);
            i.putExtra("prioridad", prioridad);
            PendingIntent pi=PendingIntent.getBroadcast(myContext, id, i,
PendingIntent.FLAG_UPDATE_CURRENT);

            Calendar AlarmCal = Calendar.getInstance();
            AlarmCal.setTimeInMillis(System.currentTimeMillis());
            AlarmCal.set(Calendar.YEAR, anyo);
            AlarmCal.set(Calendar.DAY_OF_MONTH, dia);
            AlarmCal.set(Calendar.MONTH, mes);
            AlarmCal.set(Calendar.HOUR_OF_DAY, hora);
            AlarmCal.set(Calendar.MINUTE, minutos);
            AlarmCal.set(Calendar.SECOND, 0);

            mgr.set(AlarmManager.RTC_WAKEUP,
                    AlarmCal.getTimeInMillis(),
                  pi);
```

Tasks.java

4.5.3.2. Tomar el control y pasarlo

Desde el uso de getBroadcast() en el PendingIntent de la alarma, el método OnAlarmReceiver tomará el control de forma periódica. Todo lo que esta clase realiza es pasar el control a nuestro servicio AppService, pasando por el método estático sendWakefulWork() de la clase WakefulIntentService.

Como se necesita pasar datos que permitan conocer la prioridad, titulo e identidad de la tarea, se hace a través del objeto Intent que recibe sendWakefulWork().

AlarmManager garantiza que el dispositivo estará despierto el tiempo suficiente para ejecutar el método onReceive de OnAlarmReceiver. Después de eso, AlarmManager no garantiza nada. Es por ello necesario que WakefulIntentService se encargue de mantener despierto el dispositivo.

Desde WakefulIntentService se lanza, en el día y hora especificado en la alarma, el método doWakefulWork de la clase AppService. En este último método se ejecuta el código correspondiente a la acción que debe tomar la alarma, según la prioridad de la tarea que ha lanzado esa alarma. La contextualización personaliza la alarma para evitar la intrusión innecesaria al usuario.

4.5.3.3. Aviso de alarmas contextualizado

Tal como se comenta en el punto 4.3.2 el aviso de las alarmas se ha diseñado según las necesidades del usuario. La prioridad de la tarea establece el nivel de molestia que esa alarma puede llegar a tener.

Ese control se lleva en la clase AppService donde se dispone de los siguientes mecanismos de aviso que se ejecutan en función de la prioridad.

- **Vibración de prioridad baja**. La propia barra de notificación tiene por defecto la siguiente vibración.

```
        notification.defaults |= Notification.DEFAULT_VIBRATE;
```

AppService.java

- **Vibración de prioridad media**. Se utiliza un patrón distinto y más complejo de vibración para que el usuario identifique una mayor importancia.

```
    long[] vibrate = {0,100,200,300};
    notification.vibrate = vibrate;
```

AppService.java

- **Vibración de prioridad alta y muy alta**. Para estas prioridades conviene avisar con mayor intensidad e insistencia al usuario. Es por ello que se juega con el factor tiempo al hacer vibrar al terminal, se utilizan vibraciones de 5 y de 10 segundos de duración según el parámetro de la función v.vibrate(5000) o v.vibrate(10000).

```
Vibrator v = (Vibrator)getSystemService(Con text.VIBRATOR_SERVICE);
v.vibrate(5000);
```

AppService.java

- **Aviso visual de muy baja, baja y media prioridad**. Se utiliza el sistema de notificaciones de la barra superior de Android. Un sistema que se efectúa en segundo plano sin requerir la atención inmediata del usuario.

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(ns);
int icon = R.drawable.notification_icon;
CharSequence tickerText = titulo;
long when = System.currentTimeMillis();
Notification notification = new Notification(icon, tickerText, when);
notification.flags=Notification.FLAG_AUTO_CANCEL;
CharSequence contentTitle = titulo;
CharSequence contentText = "La tarea ha vencido.";
Intent notificationIntent = new Intent(this, Tasks.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
notificationIntent, 0);
notification.setLatestEventInfo(context, contentTitle, contentText,
contentIntent);
mNotificationManager.notify(id, notification);
```

AppService.java

Los parámetros que se le indican a un NotificationManager son el icono que existe como un recurso png y el título de la tarea. Nos interesa que la notificación se ejecute en ese mismo instante que se lanza el servicio AppService, puesto que ya se ha gestionado con el AlarmManager cuándo se necesita entrar en AppService. Un parámetro muy importante es FLAG_AUTO_CANCEL. Con él se especifica que al pulsar en el aviso que se muestra en la Figura 30 se oculta, y se abre la aplicación en su actividad principal.

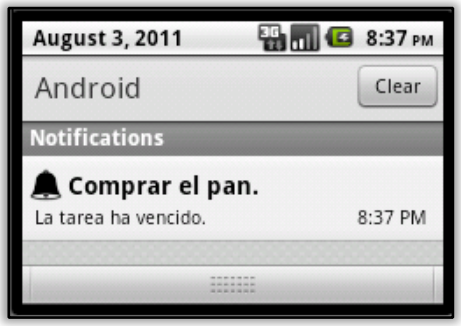

Figura 30 - Barra de notificación extendida.

- **Aviso visual de alta y muy alta prioridad**. Este tipo de aviso requiere que el usuario confirme de forma activa el dialogo que se le muestra. Es por ello que se realiza en un primer plano por encima de cualquier acción que se esté realizando.

Figura 31 - Diálogo de aviso de tarea vencida.

Existe un problema principal en la creación del dialogo estándar de Android utilizando la clase AlertDialog en la aplicación. El AlertDialog necesita el contexto de una actividad asociada, eso implica que para mostrar el diálogo debe abrirse primero la aplicación en un primer plano. Si el usuario por ejemplo está escribiendo un correo electrónico se abriría la aplicación de gestión de tareas, y después el dialogo. Sin embargo, debe ser suficiente

avisarle de forma activa con este diálogo sin necesidad de abrir la aplicación en un primer plano, y su acción se debe limita a hacer clic al botón de confirmación. Para ello en lugar de utilizar la clase AlertDialog se ha diseñado una clase con un diseño de vistas que imita el diálogo estándar, con el fondo trasparente y el botón de confirmación.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/dialog_text1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="La siguiente tarea ha vencido:"
        android:layout_marginLeft="10dip"/>
    <TextView
        android:id="@+id/dialog_text2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text=""
        android:layout_below="@id/dialog_text1"
        android:layout_marginLeft="10dip"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/dialog_text2"
        android:layout_centerHorizontal="true"
        android:text="De acuerdo"/>
</RelativeLayout>
```

/res/layout/dialogview.xml

Desde la clase MyDialog se le da valor al título de la tarea que aparece en el cuadro de diálogo, y se añade al botón el evento *onClick* con la acción finish() que cierra su actividad.

Para lanzar esta actividad desde el servicio AppService es importante el uso del parámetro FLAG_ACTIVITY_NEW_TASK. Con esto se lanza la actividad del diálogo de forma individual, sin que sea necesario abrir la actividad principal de la aplicación de gestión de tareas.

```
Intent dialog = new Intent(getBaseContext(), MyDialog.class);
dialog.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
…
getApplication().startActivity(dialog);
```

AppService.java

- **Aviso acústico de alta prioridad**. Con el aviso de sonido se completa el rango de avisos. Solo está disponible para una prioridad mayor a la prioridad media.
  Se ha personalizado el tipo de sonido añadiendo el recurso /res/raw/sound.ogg, un fichero que contiene el sonido "*BEEP*" que queremos. Su reproducción se ejecuta como sigue:

```
MediaPlayer mp = MediaPlayer.create(context, R.raw.sound);
mp.start();
```

AppService.java

- **Aviso acústico con voz de muy alta prioridad.** La librería Text-To-Speech (TTS) permite añadir textos leídos en los avisos de la aplicación. Se le envía como parámetro una cadena de texto a esta librería que convertirá en texto y la reproducirá hablando al usuario.

  Como este aviso hablado va asociado al dialogo, desde MyDialog se lanza un objeto Intent que da comienzo al servicio de la clase MySpeech.

```
…
protected void alertaSpeech(String titulo, int prioridad){
        Intent serviceIntent = new Intent();
        serviceIntent.putExtra("titulo", titulo);
        serviceIntent.setAction("com.android.tasks.MySpeech");
        myContextM.startService(serviceIntent);
}
```

MyDialog.java

  MySpeech extiende a Service, por lo tanto es un servicio que se ejecuta en un segundo plano por un periodo de tiempo indefinido. Es necesario para que no interfiera en las acciones del usuario. En el método onStartCommand del servicio MySpeech se ordena reproducir al objeto de la librería TTS.

```
public int onStartCommand(Intent intent, int flags, int startId) {
    String textToSpeech = intent.getStringExtra("titulo");
    textToSpeech = "Pending tasks. " + textToSpeech;
    mTts.setLanguage(Locale.ENGLISH);
    mTts.speak(textToSpeech, TextToSpeech.QUEUE_FLUSH, null);
    return START_STICKY;
}
```

MySpeech.java

Como se requiere que el servicio continúe ejecutándose hasta que se pare explícitamente, se devuelve la variable START_STICKY.

4.5.3.4 Estructura de datos de alarmas

Como se ha visto, una alarma se construye con un objeto de tipo AlarmManager. Este objeto inicia una alarma lanzada al sistema Android, que es quien la gestiona y se asegura de hacer cumplir su función.

Para ello se utiliza el método AlarmManager.set(), al que se le envía como parámetros un objeto PendingIntent, que se construye a partir de un objeto Intent concreto. Es posible cancelar esa alarma mediante el método AlarmManager.cancel() con el parámetro del PendingIntent que identifica a la alarma. Se observa por tanto que es necesario reconstruir el PendingIntent cada vez que se desea eliminar una alarma. Para poder gestionar más de una alarma se ha diseñado una estructura de

datos que nos permitan llevar un control de las alarmas lanzadas, y poder reconstruirlas con facilidad para su cancelación en caso necesario.

Clase NodoAlarma

Un objeto de tipo NodoAlarma representa una alarma individual. Es el elemento mínimo que contendrá la colección de datos. Se considera pues, que una alarma es un objeto de tipo NodoAlarma cuyos atributos son:

- AlarmManager. Mediante su método .set programa la alarma.
- PendingIntent. Encapsula y describe un objeto Intent. Cuando la alarma vence este Intent es lanzado como emisión de anuncio Broadcast y ejecuta la actividad de la clase destino del Intent si no se está ejecutando.
- Id. El identificador de la tarea que tiene asociada esa alarma.

Clase GestionAlarmas - ArrayList

Para gestionar varios NodoAlarma se ha implementado una clase GestionAlarmas cuya variable principal es un objeto ArrayList de tipo NodoAlarma.

```
public class GestionAlarmas {
        protected static ArrayList<NodoAlarma> al;
…
```

GestionAlarmas.java

La clase ArrayList (de la librería java.util) es un objeto de tipo lista. Esta clase permite de forma dinámica contener objetos de cualquier tipo, incluso duplicados.

Inserción de elementos a la lista

Para insertar una alarma lanzada se inicializa el objeto ArrayList y se utiliza su método .add. Anteriormente se ha creado el objeto NodoAlarma con los tres parámetros que se utilizan para programar la alarma, y este objeto es el que se añade posteriormente al ArrayList como se ve:

```
        …
        ga = new GestionAlarmas();
        …
        NodoAlarma na = new NodoAlarma(mgr,pi,id);
        ga.insertarAlarma(na);
```

Tasks.java

```
public GestionAlarmas(){
     al = new ArrayList<NodoAlarma>();
}
protected boolean insertarAlarma(NodoAlarma na){
        int indexEncontrado = GestionAlarmas.al.indexOf(na);
        if(indexEncontrado == -1){
                al.add(na);
                return true;
```

```
        }else{
                return false;
        }
}
```

GestionAlarmas.java

Por seguridad se ha controlado que la inserción no incluya elementos duplicados ya que una alarma debe ser única. Para buscar un objeto en la lista se hace uso del método indexOf() que devuelve la posición donde se encuentra el objeto, o el valor -1 si no se encuentra.

Eliminación de elementos de la lista

Cuando es necesario eliminar una alarma se debe cancelar del sistema Android, es decir, desprogramarla, y además eliminar el objeto que la registra en la lista del objeto ArrayList.

Para identificar si una alarma ha sido lanzada, o no, se busca en la lista mediante indexOf() del objeto ArrayList. Como este método funciona basándose en el método .equals del objeto, y es necesario diferenciar si una alarma es distinta a otra según el id de su tarea, se ha implementado el método .equals en NodoAlarma. Así se concreta si un NodoAlarma es igual a otro según el ID de la tarea a la que pertenecen, ya que no hay dos iguales.

```
    public boolean equals(Object nodoalarma){
                int ide = ((NodoAlarma) nodoalarma).getId();
                if (this.id == ide)return true;
                else return false;
        }
```

NodoAlarma.java

Después de encontrar el objeto NodoAlarma de la alarma que se requiere cancelar en la lista, se consultan todos sus atributos para rearmar la alarma, y así cancelarla. Además, después se elimina dicho nodo de la lista.

```
        protected boolean cancelarYeliminarAlarma(int indice){
                Object nodoTEMP = new NodoAlarma(null, null, indice);
                int indexEncontrado = al.indexOf(nodoTEMP);
                //int indexEncontrado=0;
                if(indexEncontrado != -1){
                        NodoAlarma na = al.get(indexEncontrado);
                        AlarmManager am = na.getAlarmManager();
                        PendingIntent pi = na.getPendingIntent();
                        am.cancel(pi);
                        al.remove(indexEncontrado);
                        return true;
                }else{
                        return false;
                }
        }
```

GestionAlarmas.java

Capítulo 5. Conclusiones

En esta sección se exponen las conclusiones a las que se ha llegado tras la realización de este proyecto de fin de carrera.

En este trabajo se ha diseñado e implementado una aplicación de gestión de tareas con el principal objetivo de aplicar los conocimientos estudiados a lo largo de la carrera y a su vez aprender un nuevo dominio, como es el de la programación para dispositivos móviles.

Se ha adquirido un amplio conocimiento de la plataforma Android, así como de sus componentes y posibilidades. Google ofrece una extensa documentación en la web de desarrolladores, y junto a sus tutoriales ha ayudado a conseguir una rápida formación.

Ha supuesto un reto además, conocer y aplicar la API de Facebook para integrar sus datos en el listado de tareas de la aplicación. A pesar de que ofrece ejemplos prácticos y documentación no ha resultado ser finalmente tan práctica e intuitiva como la de Google.

Por otro lado, se ha introducido el concepto de la utilización del contexto del usuario en las aplicaciones. Una funcionalidad que comienza a ser explotada en otro tipo de aplicaciones en dispositivos móviles y que en el ámbito de gestión de tareas todavía no se ha visto reflejada.

En este proyecto, el contexto ha debido ser acotado ya que tal como se muestra en la siguiente sección de trabajos futuros, se trató en un principio de ofrecer al usuario un contexto de localización geográfica, con tareas con parámetros de localización y con la integración de mapas de Google Maps en la aplicación. Finalmente se ha optado por tener en cuenta el nivel de molestia como caso práctico de aplicación del contexto en las alarmas según la prioridad de la tarea.

Cabe destacar además, la dificultad añadida de que Android puede no lanzar las alarmas que se programen en el caso de que entre en estado de espera. Esto significa que para ahorrar recursos energéticos Android suspende y pone en pausa algunos hilos de ejecución y procesos cuando no se está manipulando el dispositivo móvil. Por ello, ha sido necesaria la utilización de un componente de terceros que mantuviera despierto el sistema y los avisos fueran según lo programado.

5.1. Trabajos futuros

La aplicación de gestión de tareas que se propone en este trabajo ha quedado acotada y resuelta en un estado suficiente que posibilita su continuación y mejora de la aplicación en los siguientes distintos aspectos.

1. **Sincronización con Google**. Un primer trabajo sería la integración de copias de seguridad en la aplicación, con los servicios de tareas y eventos que ofrece Google con Google Tasks y Google Calendar respectivamente. La mayoría de aplicaciones estudiadas en el capítulo 3 disponían de esta característica, que finalmente se ha decidido no implementar para acotar el trabajo a funcionalidades que se deseaba poner en práctica para diferenciar la aplicación.

2. **Localización geográfica de tareas**. En segundo lugar, se comenzó el diseño de la aplicación con unas tareas que incluían la latitud y longitud de una localización geográfica como puede observarse en la implementación de la base de datos. Finalmente no se integró Google Maps en la aplicación por el mismo motivo de las copias de seguridad. Se ha optado por acotar el proyecto.

3. **Filtrado de categorías en la pantalla principal**. Un tercer posible futuro trabajo es el filtrado de tareas según su categoría en la lista principal de la aplicación. Esta funcionalidad se comenzó a implementar como actividad de la clase FilterTasks.java, que listaba las categorías disponibles con casillas de verificación para visualizarlas u ocultarlas en la actividad principal de la aplicación.

4. **Programación de widgets**. Un cuarto trabajo es la creación de un elemento visual gráfico para el escritorio, llamado widget. Todas las aplicaciones estudiadas en el punto 3 disponen de widgets. Es realmente útil en este tipo de aplicaciones tener lo más visible posible una agenda con recordatorios visuales de las tareas a realizar, por ejemplo.

5. **Realización de pruebas con el usuario final**. Por último, el quinto trabajo a realizar es la realización de pruebas junto a usuarios finales. De esta forma, conociendo sus opiniones y analizando cómo la aplicación es utilizada, pueden detectarse problemas de usabilidad a solventar o nuevas funcionalidades a cubrir.

Referencias

1. http://www.cesareox.com/opinion/articulos/94575/las_listas_de_tareas.html
2. http://appleweblog.com/2011/03/por-que-ios-es-mas-fluido-que-android
3. http://developer.apple.com/programs/ios/
4. http://es.wikipedia.org/wiki/Plataforma_de_software
5. http://es.wikipedia.org/wiki/Distribuci%C3%B3n_de_software
6. http://es.wikipedia.org/wiki/Dalvik
7. http://es.wikipedia.org/wiki/Open_Handset_Alliance
8. http://developer.android.com/guide/basics/what-is-android.html
9. http://developer.android.com/resources/dashboard/platform-versions.html
10. http://es.wikipedia.org/wiki/Android
11. http://es.wikipedia.org/wiki/Apple_iOS#Licencia
12. http://es.wikipedia.org/wiki/Objective-C
13. http://android-so.com/que-es-android-historia-y-caracteristicas-del-sistema-operativo
14. http://tech.fortune.cnn.com/2011/06/21/needham-androids-market-share-peaked-in-march/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+fortuneapple20+%28FORTUNE%3A+Apple+2.0%29
15. https://github.com/commonsguy/cwac-wakeful/blob/master/README.markdown
16. http://blog.nielsen.com/nielsenwire/?p=28628
17. http://es.scribd.com/doc/29737789/Android-Content-Provider-tutorial
18. http://developer.android.com/guide/topics/providers/content-providers.html
19. http://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas
20. Hello Android (3rd edition): Introducing Google's Mobile Development Platform by Ed Burnette
21. Towards a Unified Knowledge-Based Approach to Modality Choice – Yulia Bachvarova, Betsy van Dijk, Anton Nijholt. Human Media Interaction Group, University of Twente. The Netherlands
22. The Busy Coder's Guide to Advanced Android Development – Mark L. Murphy
23. Personalization for unobstrusive service interaction. Miriam Gil, Pau Giner, Vicente Pelechano. Springer-Verlag London Limited 2011
24. http://www.aleydasolis.com/seo/el-futuro-del-seo-las-redes-sociales/
25. http://developers.facebook.com/docs/guides/mobile/#android
26. http://developers.facebook.com/docs/reference/api/event/
27. http://developer.android.com/guide/topics/ui/dialogs.html
28. http://developer.android.com/resources/tutorials/views/index.html
29. https://market.android.com/details?id=com.greenbeansoft.CheckmarkLite&feature=search_result
30. https://market.android.com/details?id=org.dayup.gtask&feature=search_result
31. https://market.android.com/details?id=jp.co.johospace.jorte&feature=search_result
32. https://market.android.com/details?id=jp.co.elecom.android.elenote&feature=search_result
33. https://market.android.com/details?id=com.timsu.astrid&feature=search_result
34. https://market.android.com/details?id=no.intellicom.tasklist&feature=search_result

35. https://market.android.com/details?id=com.als.taskstodo&feature=search_result
36. https://market.android.com/details?id=com.mikesandroidworkshop.android.tasklist&feature=search_result
37. https://market.android.com/details?id=com.taskos
38. http://buscon.rae.es/draeI/SrvltGUIBusUsual?LEMA=tarea&origen=RAE&TIPO_BUS=3
39. http://buscon.rae.es/draeI/SrvltGUIBusUsual?LEMA=evento&origen=RAE&TIPO_BUS=3
40. http://es.wikipedia.org/wiki/SQLite

Fotografías utilizadas en las categorías de la aplicación.

- 1965 Stephen's Toy Cars, de Serendigity**.** Según licencia CC BY-SA 2.0.
- Airbus A380, de Peter Pearson. Según licencia CC BY-SA 2.0.
- Apple Mouse, de Jose Carlos Castro. Según licencia CC BY-NC 2.0.
- Birthday Cake – Candles, de Jessica Diamond. Según licencia CC BY-SA 2.0.
- Book, de Jason Gulledge. Según licencia CC BY 2.0.
- House???, de Cindy Seigle. Según licencia CC BY-NC-SA 2.0.
- Money, de Mait Jüriado. Según licencia CC BY-NC-SA 2.0.
- Music, de Theoddnote. Según licencia CC BY 2.0.
- Square made from a group of tennis balls, Horia Varlan. Según CC BY 2.0.
- Suit 1, de Nelson Pavlosky. Según CC BY-SA 2.0.