Universitat Politècnica de Catalunya
ETSETB

Proyecto Final de Carrera

Diseño e implementación de una aplicación móvil bajo la plataforma Android para la colaboración en la creación de historias

Autor: Jalal Souky Salha
Tutor Académico: Josep Paradells Aspas

Octubre 2011

Contenido

Índice de Figuras

Índice de Tablas

Acrónimos

3G – Tercera Generación.
MVC – Comunidad Virtual Móvil.
MTG – Music Technology Group.
OS – Sistema Operativo.
UI – Interfaz de Usuario.
GUI – Interfaz Gráfica de Usuario.
API – Interfaz de Programación de Aplicaciones.
XML – eXtensible Markup Language.
ANR – Application Not Responding.
SDK –Kit de Desarrollo de Software.
VM – Máquina Virtual
AVD – Dispositivo Virtual Android
ADB – Puente de Depuración Android
NMEA – National Marine Electronics Association
UDP – User Datagram Protocol
TCP – Transmission Control Protocol
RFC – Requests For Comments
SSL – Secure Socket Layer
TLS – Transport Layer Security
WAVE – Waveform Audio Format
RIFF – Resource Interchange File Format

REST – Representational State Transfer

IFF – Interchange File Format

Resumen

En este proyecto se presentará el diseño e implementación de *Tales*; una aplicación móvil bajo la plataforma Android para la colaboración en la creación de historias. A través de una interfaz gráfica atractiva e intuitiva los usuarios podrán, mediante de un modelo cliente-servidor, escuchar la participación de los personajes en una historia y tendrán la posibilidad de darle continuidad y agregar nuevas partes al relato. Cada uno de los personajes posee un tipo de transformación de voz y un avatar asociado, lo que les brindará un carácter específico y mayor identidad. Por otra parte, se tratarán aspectos técnicos de la plataforma Android mostrando sus características principales y las funcionalidades que hacen posible la implementación de aplicaciones móviles de este tipo. Adicionalmente se presentarán aspectos generales del negocio llegando a conclusiones importantes referentes al creciente mundo de las aplicaciones móviles.

Palabras clave: Android, aplicaciones móviles, colaboración, cliente-servidor.

Capítulo 1
Introducción

En este capítulo primero se expondrán las razones que llevaron al desarrollo de este proyecto, tanto las implicaciones académicas de futuros campos de investigación, como el potencial profesional y económico que conlleva la realización de un proyecto de este tipo. Así mismo, se explicará la visión detrás de este proyecto como parte del Grupo de Investigación en Tecnología Musical, o MTG por sus siglas en inglés, de la Universidad Pompeu Fabra. También se presentarán los objetivos que este proyecto pretende cumplir, al igual que los aspectos que se escapan de su alcance. Finalmente se explicará la estructura de este documento a partir de la finalización del presente capítulo.

1.1 Motivación

En los últimos años el mundo ha sido testigo de una proliferación de dispositivos móviles inteligentes o *smartphones* y de redes inalámbricas. Esta proliferación se complementa con los avances en el poder computacional de los dispositivos y en el aumento de ambientes de software que estos soportan. Aunque considerable, no es sorpresa que las ventas de *smartphones* hayan aumentado en un 48,7% en el primer cuarto de 2010 respecto al mismo período del año anterior[35]. Otros estudios muestran que si este aumento sigue según las proyecciones, los smartphones pasarán a ser el primer referente en dispositivos móviles para finales de 2011[61]. Adicionalmente, estos dispositivos son

cada vez más asequibles para el usuario general y los servicios de tercera generación (3G) por parte de las compañías de telefonía móvil son cada vez más comunes.

Con esta proliferación de dispositivos móviles y redes inalámbricas, los usuarios esperan poder llevar a cabo sus roles en las comunidades virtuales en cualquier lugar que se encuentren durante su día a día. También, nuevos entornos virtuales se crean sin dar evidencia que en algún momento pararán. Por ello, el nacimiento de una nueva forma de comunidad virtual con dispositivos móviles es una manifestación natural del contexto tecnológico expuesto anteriormente. Estas comunidades son conocidas como Comunidades Virtuales Móviles (MVCs, Mobile Virtual Communities) y comienzan a ser un prometedor campo de investigación.

Este crecimiento también se puede ver reflejado en el mercado de aplicaciones móviles y en el compromiso de las empresas en proporcionar este tipo de servicios. Sólo en los Estados Unidos el mercado de aplicaciones móviles esperaba obtener $3800 millones en ventas en 2007, con proyecciones de hasta $9000 millones para el 2011[14]. Dicho crecimiento en el mercado viene de la mano con un aumento de inversión por parte de empresas de diferentes industrias y entidades gubernamentales.

Las implicaciones tecnológicas son igualmente prometedoras ya que se espera una diversificación en el tipo de aplicaciones que se desarrollarán, abarcando sectores como la salud y las herramientas para la gestión profesional. Sectores que anteriormente se veían opacados anteriormente por aplicaciones de carácter lúdico.

1.2 Definición del Proyecto

Este proyecto se desarrolló en el Grupo de Investigación en Tecnología Musical (MTG, Music Technology Group) de Universidad Pompeu Fabra en Barcelona. Este grupo se especializa en las tecnologías digitales para el procesado, descripción y modelado de audio y música. Un aspecto a destacar es el carácter interdisciplinario del grupo, que combina competencias del Procesado de Señales, el Aprendizaje Automático y la Interacción Hombre Máquina. El MTG siempre ha buscado la mejora de las tecnologías de la información y las comunicaciones en el ámbito tecnológico y social de la música y el sonido.

Dentro de su tarea investigativa, el MTG posee un grupo destinado al proyecto Music 3.0. Este es un proyecto de I+D financiado por el programa Avanza Contenidos del Ministerio de Industria, Turismo y Comercio de España con el principal objetivo de desarrollar un sistema web para la creación, interacción y socialización de la música. Esta iniciativa busca integrar tecnologías recientes de la Web 2.0, herramientas on-line avanzadas y grandes repositorios de información. La visión detrás del concepto del Music 3.0 se puede apreciar en la Figura 1.1 que muestra a esta iniciativa como el punto de encuentro de herramientas de producción, comunidades de colaboración y la gestión de contenido.

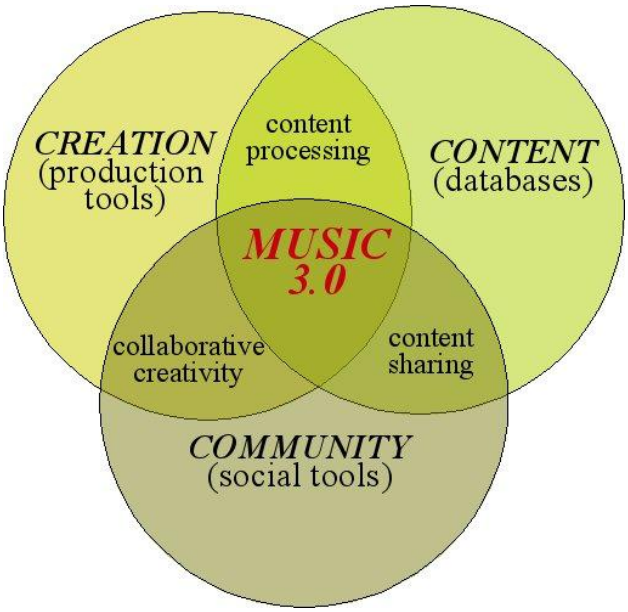

Figura 1.1. Diagrama conceptual del proyecto Music 3.0[47].

El proyecto expuesto en esta memoria busca la integración de tecnologías desarrolladas anteriormente por el MTG y que no han sido integradas a su máximo potencial, con la finalidad de obtener una herramienta que facilite crear y compartir nuevo contenido a través de medios innovadores siguiendo la filosofía del Music 3.0.

En un principio no se fijaron parámetros específicos, tanto creativos como tecnológicos, para la creación de este proyecto. Tras varias propuestas de diversa índole, se decidió desarrollar una aplicación móvil bajo el Sistema Operativo (OS) Android que sirviera como principal herramienta de colaboración para una MVC de cuenta cuentos llamada *Tales*, o *Cuentos* en inglés. Esta idea fue concebida para formar parte de los talleres académicos

de la Conferencia Internacional de Procesado de Sonido y Música que se llevó a cabo en Barcelona en Julio de 2010.

La aplicación móvil de Tales permite a los integrantes de esta MVC participar de manera pasiva o activa, desde sus dispositivos móviles Android, en historias creadas anteriormente por otros usuarios. Estas historias no se desarrollan mediante la escritura de diálogos y narraciones, sino a través de herramientas audiovisuales para la creación de las mismas haciendo a la aplicación innovadora y más atractiva la experiencia general del usuario.

1.2.1 Objetivos Generales

El objetivo principal de este proyecto es el diseño y desarrollo de una versión alpha de Tales: una aplicación móvil bajo el OS de Android, que permita a los integrantes de la MVC bajo el mismo nombre colaborar en la creación de historias a través de contenido audiovisual.

1.2.2 Objetivos Específicos

1. Aprender el lenguaje de programación y tecnologías utilizadas para el desarrollo de aplicaciones móviles Android.
2. Mostrar algunas de las capacidades de la plataforma Android para el desarrollo de aplicaciones móviles.
3. Proponer y desarrollar funcionalidades adicionales para aplicación.
4. Comprender los elementos necesarios de una Interfaz Gráfica de Usuario (GUI) atractiva e intuitiva.
5. Hacer un estudio de los costes para el desarrollo de una aplicación móvil como Tales.

1.3 Alcances y Limitaciones

- *Tales* busca mostrar el potencial de Android como infraestructura de desarrollo de aplicaciones móviles.

- El proyecto integrará tecnologías propias del MTG siguiendo la visión de trabajo de Music 3.0.
- El usuario móvil podrá obtener metadata y escuchar las historias, de igual manera que podrá grabar y transformar sus propios diálogos y narraciones en historias preexistentes.
- La aplicación será únicamente una herramienta móvil para la colaboración y no será independiente de la plataforma web que la soporta. La aplicación mostrada debe ser considerada como una versión de alpha del proyecto, dando cabida a amplias mejoras y posteriores validaciones.
- Esta aplicación no debe ser pensada como un producto final listo para su publicación. Este proyecto sólo muestra una versión de demostración de la aplicación y el potencial de Android como plataforma para su desarrollo.

1.4 Estructura del Documento

A continuación se presentará un breve resumen del contenido y estructura de las memorias del proyecto. Se mencionarán los aspectos más importantes de cada uno de los cinco capítulos que comprenden este documento.

En el capitulo siguiente se mostrará el Estado del Arte donde se explicarán de manera detallada todos los conceptos necesarios para comprender en su plenitud las tecnologías y protocolos utilizados en el desarrollo de este proyecto, así como el modelo de comunicación empleado en este tipo de aplicaciones. Se prestará especial atención a los elementos de la plataforma Android.

En el capítulo 3, Actividades de la Aplicación, se presentarán separadamente cada una de las actividades de corresponden a cada una de las pantallas de Tales. Este capítulo servirá al lector como manual de la aplicación ya que explicará cada uno de los elementos que el usuario encontrará a medida que interactúa con el móvil y cuáles son sus funciones.

Después de describir qué hace la aplicación, se pasará al capítulo 4 donde se explica de manera detallada cómo lo hace. Se hará con un enfoque similar al del capítulo anterior de

manera que estará separado por pantallas, pero en este caso se explicarán las clases y procedimientos en el código y su interacción.

Finalmente, en el capítulo 5 se presentarán las conclusiones del proyecto, tanto particulares a la aplicación como a la plataforma Android. Adicionalmente se exhibirán los objetivos alcanzados y las mejoras que aún quedan por realizar, dando entrada a preguntas abiertas y futuras líneas de investigación.

Capítulo 2
Estado del Arte

En este capítulo se presentarán en detalle todos aquellos conceptos teóricos que el lector deberá conocer para comprender los aspectos de implementación de este proyecto que se tratarán en el capítulo 4. Se explicarán algunos conceptos generales sobre las Interfaces de Programación de Aplicaciones, así como se entrará en mayor detalle en los aspectos relacionados concretamente con la plataforma Android. Posteriormente, se presentarán todos aquellos entornos de desarrollo que fueron necesarios para la implementación de Tales, así como otras aplicaciones de este tipo. Finalmente se explicarán los protocolos y tecnologías que componen el contexto científico de este proyecto.

2.1 Interfaz de Programación de Aplicaciones

Una Interfaz de Programación de Aplicaciones o API (en inglés, Application Programming Interface) es una interfaz implementada por un programa de software, utilizada para interactuar con otro programa de software. Una API es implementada por aplicaciones, librerías y protocolos para establecer el esquema de los parámetros de solicitud y respuesta de los servicios que ofrecen.

Las API pueden ser generales, si están totalmente incluidas en las librerías de un lenguaje de programación, o específicas, si están concebidas para llevar a cabo una tarea concreta. Una API también puede ser clasificada como dependiente del lenguaje, si únicamente puede ser empleada con la sintaxis de un lenguaje de programación en específico, o independiente del lenguaje, si está implementada para utilizarse con cualquier lenguaje de programación. Esta última característica es deseable en el caso de APIs orientadas a servicios que no estén vinculadas a un sistema o proceso específico y puedan ser previstas como servicios web[64].

2.1.1 Aplicaciones cliente-servidor

Para poder entender la manera en la que la aplicación *Tales* funciona, es necesario comprender algunos conceptos básicos en el modelo de comunicación cliente-servidor. Este es un modelo lógico donde un programa de software, el cliente, realiza peticiones a otro programa, el servidor, el cual responde. Como se acaba de mencionar, este tipo de modelo es simplemente lógico donde se separan las responsabilidades de computación con el fin de orgamizar y facilitar el diseño, implementación y gestión del sistema. Teniendo esto en cuenta, es importante destacar que el servidor no es necesariamente un solo programa o una sola máquina donde las funciones de *esclavo* pueden estar distribuidas en una red [65].

Por lo general la comunicación entre el cliente y el servidor se realizan sobre TCP/IP donde ambas partes se pueden identificar la una a la otra y donde pueden establecer un canal de comunicación. En capas superiores, donde los programas de software responsables de los servicios de la aplicación se comunican, se hacen uso de APIs para establecer las reglas de la comunicación y el formato en el cual se deben presentar los datos de los mensajes y peticiones.

Por otra parte, en el otro lado del cliente, se encuentra presente otra interfaz. En este caso, es la Interfaz Gráfica de Usuario, o GUI por sus siglas en inglés, utilizada en la interacción entre el usuario final y el cliente. Esta interfaz se basa en recursos como vistas y botones que permiten al usuario final comunicarse con el cliente de una manera que ambos entiendan.

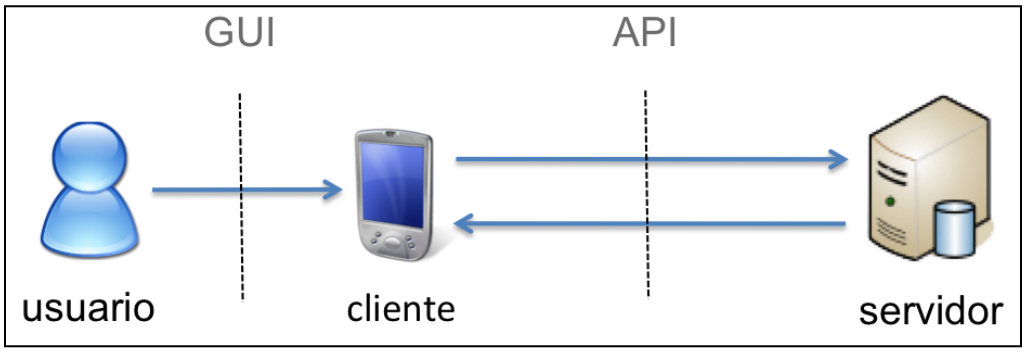

Figura 2.1. Arquitectura básica de la aplicación Tales.

En la figura 2.1 se muestra un diagrama simplificado de lo que se acaba de explicar y, aunque existen diferentes modelos de cliente-servidor, esto el lo único necesario para comprender de manera general la arquitectura lógica de la aplicación de Tales y su comunicación con los diferentes servicios remotos que utiliza.

2.2 Android

Android es una plataforma para dispositivos móviles desarrollada por la Open Handset Alliance, que incluye un sistema operativo, middleware y un conjunto de aplicaciones clave[7]. Estas aplicaciones comprenden las funciones básicas de cualquier dispositivo móvil, como pueden ser el gestor de llamadas y mensajería SMS y el gestor de correo electrónico. Android se basa en el kernel de Linux versión 2.6 como núcleo para los servicios de sistema como la seguridad, gestión de memoria, gestión de procesos, funciones de red, entre otros. El kernel también actúa como capa de abstracción entre el hardware y el resto de la pila de software, y posee una gran cantidad de librerías especiales para Android. El lenguaje de desarrollo de aplicaciones es al ampliamente conocido Java de Sun Microsystems. Android también posee su propia máquina virtual llamada Dalvik Virtual Machine, concebida especialmente para dispositivos móviles[23]. Estos temas serán tratados con mayor profundidad en las secciones siguientes de este capítulo.

Los prometedores aspectos tecnológicos de Android también se ven bien reflejados en las ventas. Según la principal compañía de estudios de mercado de los Estados Unidos, NPD Group, Android obtuvo el primer lugar en ventas de *Smartphones* durante el segundo cuarto de 2010 con un 33% del mercado Americano, seguido en segundo lugar por

BlackBerry OS con 28% y en tercer lugar iOS de Apple con un 22%[62]. Por otra parte, según cifras publicadas por Canalys, una respetada firma analísta internacional, posiciona a Android como la plataforma móvil más utilizada en el mundo desde el tercer cuarto de 2010 con un 33% del mercado, seguida por Symbian de Nokia con un 30%, luego iOS con un constante 16% y RIM con poco más del 14% [22].

2.2.1 Inicios de la plataforma Android

Los dispositivos móviles usan una gran variedad de sistemas operativos propietarios como Symbian OS, Windows Mobile de Microsoft, Mobile Linux, iOS de Apple, Moblin de Intel, entre otros. Aunque también existen algunos sistemas operativos móviles de código abierto como OpenMoko[51] y LiMo[46], no poseen el suficiente respaldo por parte de fabricantes y proveedores de servicio. Hasta entonces ningún sistema operativo se había posicionado como el estándar a seguir. Adicionalmente, las APIs y los ambientes de desarrollo para aplicaciones móviles eran muy restrictivos, haciendo la tarea de los desarrolladores difícil y dispersa. Este es el contexto en el que entra Android, buscando ser el estándar dentro de la amplia y confusa gama de sistemas operativos para dispositivos móviles. Para ello, Android promete crear una plataforma abierta, asequible, libre y avanzada para dispositivos móviles y el desarrollo de sus aplicaciones[40].

En Julio de 2005, Google compra casi de manera secreta a Android, Inc., una nueva compañía de tan sólo 22 meses dedicada al desarrollo de aplicaciones móviles. Según fuentes no oficiales de la revista BusinessWeek [21], se presumía que esta *startup* estaba trabajando en un nuevo sistema operativo para teléfonos móviles. Esta compra también asimiló al principal capital humano de Android, Inc., quienes pasaron a formar parte de la plantilla de Google para liderar este proyecto.

El 5 de Noviembre de 2007, y tras numerosas especulaciones por parte de los medios, la Open Handset Alliance, un consorcio de varias compañías que incluye a Google, HTC, Texas Instruments, Intel, LG, Motorola, Broadcom Corporation, Marvell Technology Group, NVIDIA, Samsung Electronics, Sprint Nextel, Qualcomm y T-Mobile fue revelado con el objetivo de desarrollar un estándar abierto para dispositivos móviles. Poco después presentó su primer producto, Android, una plataforma creada bajo el kernel de Linux versión 2.6[50].

Menos de un año después, el 21 de Octubre de 2008, Google publicó todo el código fuente de la plataforma bajo una Lincencia Apache[20], con la cual proveedores pueden agregar extensiones propietarias sin tener que presentarlas a esta comunidad de código abierto. Desde entonces, la plataforma Android ha presentado cuatro versiones y se encuentra trabajando en otras dos, Gingerbread[59] y Honeycomb[15], cuyos lanzamientos están estipulados para finales de 2010 y durante 2011 respectivamente.

2.2.2 Características Principales de Android

En esta sección se presentarán las características más destacables de la plataforma Android y sus diferencias respecto a otros sistemas operativos móviles mencionados anteriormente. Dentro de estas características se pueden mencionar las siguientes[7]:

- Entorno de aplicaciones que permite la reutilización y sustitución de componentes.
- Maquina virtual Dalvik, optimizada especialmente para dispositivos móviles.
- Buscador integrado basado en el motor de búsqueda de código abierto WebKit.
- Almacenamiento de datos estructurado a través de SQLite.
- Librerías personalizadas para la optimización de gráficos 2D y soporte para gráficos 3D basados en OpenGL ES 1.0.
- Soporte para diferentes formatos de audio, video e imágenes (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF, entre otros).
- Ambiente de desarrollo completo que incluye emulador del dispositivo, herramienta para depuración, perfiles de memoria y rendimiento, y un módulo para Eclipse IDE.

El entorno abierto de aplicaciones de Android ofrece a los desarrolladores la posibilidad de crear aplicaciones avanzadas e innovadoras, permitiendo hacer uso de las funciones del hardware del dispositivo, acceder a información de localización, ejecutar procesos en diferentes hilos, entre otras cosas. Los desarrolladores también tienen acceso a las mismas APIs utilizadas en las aplicaciones clave. Por otra parte, la arquitectura de las aplicaciones está diseñada con el fin de simplificar el reutilización de componentes, de manera que cualquier aplicación puede publicar sus funcionalidades para que otra aplicación pueda hacer uso de ellas. Este mecanismo permite que los componentes sean remplazados por el usuario.

Una de las principales características de Android es la Dalvik Virtual Machine, creada para utilizarse en teléfonos móviles, Tablet PCs y netbooks, en lugar de la comúnmente utilizada Java Virtual Machine de Sun. La principal diferencia entre estas dos máquinas virtuales es que Dalvik utiliza una arquitectura basada en registros mientras que la máquina virtual de Java se basa en pilas. Aunque los méritos de estos dos tipos de máquinas virtuales se encuentran en constante debate[58], la Dalvik Virtual Machine se destaca por su capacidad de trabajar en sistemas con poca memoria, lo que le permite correr en sistemas embebidos y funcionar correctamente en casos de bajo suministro eléctrico. Adicionalmente, Dalvik fue concebida de manera que el dispositivo pueda ejecutar eficientemente diferentes instancias de la máquina virtual. Esto permite que cada aplicación corra en su propio proceso y en su propia instancia de la máquina virtual. También, ejecuta los archivos en su propio formato, Dalvik Excecutable (.dex), el cual entre otras cosas también optimiza el uso de memoria.

Otra característica que vale la pena mencionar es el uso de SQLite como gestor bases de datos relacionales. Este sistema reduce la latencia en el acceso de la base de datos y está contenido en una relativamente pequeña biblioteca en el lenguaje C[73]. Además de esto posee un diseño simple lo cual lo hace apto para dispositivos móviles.

La plataforma Android incorpora un motor gráfico 2D en SGL que mejora la eficiencia de los algoritmos gráficos. Para la implementación de gráficos 3D la plataforma cumple con la especificación OpenGL ES 1.1 el cual corresponde a la versión OpenGL 1.3.

Adicionalmente la plataforma Android otorga soporte a las funciones de hardware que se pueden encontrar en la mayoría de dispositivos móviles. Entre estas podemos destacar:

- Telefonía GSM.
- Soporte para Bluetooth, EDGE, 3G y WiFi.
- Cámara integrada, GPS, compás y acelerómetro.

Todas estas características vienen acompañadas por un ambiente de desarrollo en constante actualización y mejora. Este ambiente permite la utilización de dichas funcionalidades de una manera eficiente y personalizada para dispositivos móviles. Los aspectos relacionados con este tema se explicarán con mayor detalle en la sección 2.3.

2.2.3 Componentes de una aplicación Android

Como se mencionó en la sección anterior, una de las características más destacables de la plataforma Android es el poder reutilizar elementos de una aplicación en otra sin ser necesario reincorporar el código de este elemento en la nueva aplicación.

El sistema debe ser capaz de iniciar el proceso de una aplicación cuando alguna parte de esta sea necesaria y crear una instancia del objeto Java de esta parte. Por ello, a diferencia de aplicaciones en otros sistemas, las aplicaciones Android no tienen un único punto de entrada. En lugar de esto, las aplicaciones poseen componentes esenciales que el sistema puede instanciar e iniciar cuando sea necesario. Existen cuatro tipos de estos componentes: *Activities*, *Services*, *Broadcast Receivers* y *Content Providers*.

Las *Activities* o actividades, representan la componente de interfaz gráfica de una tarea con la que el usuario interactúa. Por ejemplo, una actividad podría ser mostrar la lista de contactos que el usuario tenga en su cuenta de correo electrónico o mostrar vistas miniatura de fotografías almacenadas en la tarjeta SD. Una aplicación para enviar mensajes de texto podría, por ejemplo, tener una actividad que muestre la lista de contactos para seleccionar el destinatario, otra que inicie un editor de texto para escribir y enviar el mensaje y otra que permita ver el historial de mensajes anteriores y editar preferencias. El número de actividades de una aplicación, dependerá únicamente de su funcionalidad y diseño.

Aunque las actividades trabajan de manera conjunta dentro de una aplicación, cada una de ellas es independiente de la otra. Normalmente, existe una actividad inicial que es presentada al usuario una vez que comienza la aplicación. La manera de moverse dentro de una aplicación, es iniciar a la siguiente actividad dentro de la actividad actual o regresar a la anterior. Esto se explicará en breve dentro de este apartado.

Las actividades poseen un ciclo de vida en el cual instancias de las mismas son almacenadas en una pila para que puedan ser recuperadas posteriormente. Esta pila gestiona el uso de la memoria del dispositivo de manera transparente al usuario el cual navega de manera fluida a través de una o varias aplicaciones. Si el dispositivo no posee

recursos ociosos y otro componente con una prioridad mayor los necesita, el sistema evalúa cuál es la actividad más prescindible y la elimina *parcialmente* del historial. Se dice parcialmente, ya que el sistema siempre guarda una copia del estado final de la actividad a pesar de haber eliminado su instancia. De esta manera si el usuario regresa a una actividad que ya no se encuentra en la pila, el sistema creará de nuevo una instancia de la misma y reanudará el último estado en el que se encontraba. En la figura 2.2 se puede observar el diagrama de flujo del ciclo de vida de una actividad.

Otro de los componentes de una aplicación Android son los *Services* o servicios. La característica principal de los servicios es que no poseen una interfaz gráfica, sino son procesos que se ejecutan en el fondo durante un período indefinido de tiempo. Una aplicación típica para este tipo de componentes podría ser un reproductor de audio, donde el usuario puede navegar en diferentes aplicaciones mientras escucha música. Es posible vincular una actividad a un servicio de manera que la actividad inicie el servicio, o también que el servicio proporcione cierta información a la actividad mientras se esté ejecutando. Siguiendo el ejemplo del reproductor de audio, una actividad que muestre una lista de canciones puede iniciar el servicio de reproducción una vez que el usuario haya escogido el álbum de su artista favorito. De la misma manera, el servicio podría dar información a la actividad para que muestre una barra que indique el progreso de la canción que está sonando en ese momento.

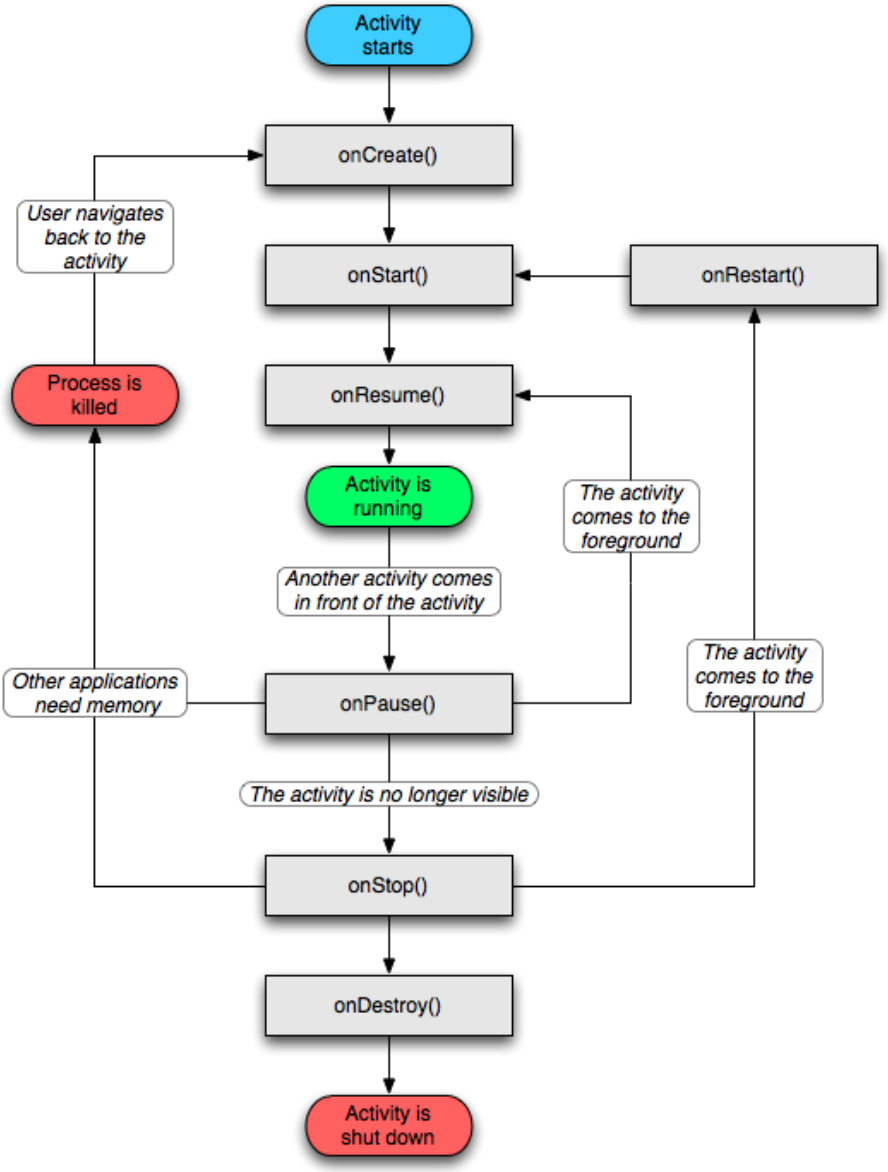

Figura 2.2. Ciclo de vida de una actividad Android[4].

También existen los *Broadcast Receiver* o receptores de broadcast, que simplemente se encargan de recibir notificaciones de broadcast por parte del sistema, como una altera de nivel bajo de batería o un cambio de uso horario. Una aplicación puede tener varios receptores y actuar de una manera determinada para cada uno de ellos. Aunque estos receptores no se manifiestan en la interfaz gráfica, pueden iniciar o modificar una actividad que se encuentre en el primer plano. Por ejemplo, una aplicación podría disminuir el brillo de la pantalla en caso que la batería esté a punto de agotarse.

Finalmente tenemos los *Content Providers* o proveedores de contenido, los cuales permiten disponer de un conjunto de datos específicos de una aplicación en otra. Esta información puede estar contenida en una base de datos en SQLite, en los archivos del sistema o en cualquier otra parte que se crea conveniente.

Estos son los cuatro componentes que pueden forman una aplicación Android. Sin embargo, existen otros elementos fundamentales que hacen posible la inicialización de componentes y la interacción de los mismos.

Las *Activities*, *Services* y *Broadcast Receivers* son iniciados por mensajes asíncronos llamados *intents*. Estos últimos pueden ser usados para iniciar una actividad, nombrar la acción anunciada por un receptor de broadcast o comunicarse con un servicio. Los *intents* permiten enlazar código de diferentes aplicaciones durante la ejecución[3]. Su uso más frecuente es en el inicio de actividades, en el que pueden ser considerados como un *pegamento* entre las mismas. Por otra parte, los proveedores de contenido son iniciados por una clase especial denominada *ContentResolver*.

Un *intent* puede explícitamente nombrar el componente que quiere iniciar. Sin embargo, Android puede localizar implícitamente el componente más adecuado para llevar a cabo la acción deseada. Para ello, el sistema compara el *intent* con los *Intent Filters* o filtros de intent para encontrar el componente indicado. Los filtros de un componente informan a Android el tipo de *intents* que este componente puede realizar.

Uno de los elementos más importantes y obligatorio en cualquier aplicación Android es el *Manifest File* o manifiesto. Antes que Android pueda iniciar un componente de una aplicación, debe saber que este componente existe. Por lo tanto, las aplicaciones declaran a sus componentes, en una estructura XML[76], eXtensible Markup Language, en su manifiesto el cual es incluido en paquete Android conjuntamente con el código de la aplicación, archivos y otros recursos.

El manifiesto tiene otras funciones como nombrar librerías adicionales que deben ser enlazadas a la aplicación e identificar los permisos que esta espera que se le otorguen.

Como se puede observar en la figura 2.3, el manifiesto especifica primeramente el nombre del paquete que contiene todos los archivos Java de la aplicación y la versión del SDK que se utilizó. También establece el nombre y la imagen del ícono que la aplicación mostrará en el menú de aplicaciones. Posteriormente se puede observar la etiqueta XML <activity> donde se indica el nombre de la actividad y los *Intent Filters* de la misma. Como la actividad mostrada en la figura representa la pantalla de inicio, se utilizó el filtro android.intent.action.MAIN.También se puede observar la etiqueta <uses-permission>, que en este caso solicita permiso para accesar a Internet.

2.2.4. Interfaz de Usuario Android

La plataforma Android dispone de dos clases Java especialmente diseñadas para la construcción de la interfaz de usuario, o UI, de una aplicación. Estas clases son *View* y *ViewGroup*.

La clase *View* funciona como unidad base para la creación de las UI y para implementar otros componentes de UI como son los *widgets*. Estos últimos son objetos predeterminados para crear botones, campos de texto, entre otras cosas. Por otra parte, la clase *ViewGroup* funciona como base para la implementación de *layouts*, los cuales ofrecen diferentes tipos de diseño que establecen la disposición de los elementos *View* en la UI. Estos pueden ser lineales, absolutos, relativos, y mucho más.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="mtg.tales">
      <uses-sdk android:minSdkVersion="3" />
      .
      .
      .
      <application
            android:icon="@drawable/icon"
            android:label="Tales"
            android:debuggable="true">
            <activity
                  android:name=".Login"
                  android:label="Login">
                  <intent-filter>
                        <action
                              android:name="android.intent.action.MAIN" />
                        <category
                              android:name="android.intent.category.LAUNCH
                        ER" />
                  </intent-filter>
            </activity>
                  .
                  .
                  .

      </application>
      <uses-permission
            android:name="android.permission.INTERNET">
      </uses-permission>
            .
            .
            .

</manifest>
```

Figura 2.3. Fragmento del ManifestFile.xml de la aplicación Tales.

Un objeto *View* puede ser definido como una estructura de datos que establece los parámetros de diseño y el contenido de un área rectangular específica dentro de la pantalla del dispositivo. Estos objetos también manejan el tamaño, diseño, enfoque y desplazamiento del área rectangular en la cual residen.

Tal y como se muestra en la figura 2.4, la UI de una actividad es esquematizada a través de un árbol jerárquico de nodos *View* y *ViewGroup*. Este árbol puede ser tan simple o complejo como el diseño de la aplicación lo exija.

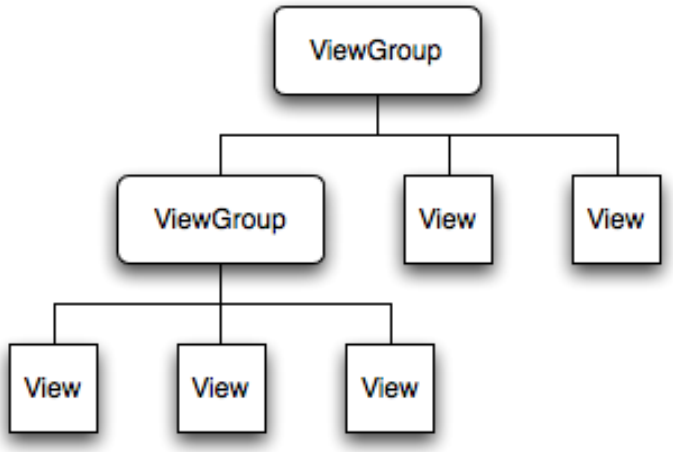

Figura 2.4. Árbol jerárquico de una UI en Android[2].

Para poder representar este árbol jerárquico en la pantalla del dispositivo, la actividad únicamente debe hacer una llamada a un método específico contenido en la clase *View* y pasar una referencia al objeto ubicado en la raíz. Luego el sistema utiliza esta referencia y los parámetros contenidos en el diseño del *layout* para hacer las mediciones y dibujar el árbol. Para ello, el nodo raíz solicita a sus nodos hijos que se dibujen a ellos mismo, a la vez que los nodos *ViewGroup* llaman a sus nodos hijos para que también lo hagan. Cada uno de los nodos solicitan un tamaño y una posición dentro de sus padres, quienes tienen la última decisión de que tamaño pueden tener y en donde se pueden ubicar. Como los elementos son dibujados de manera jerárquica, es decir, desde la raíz hasta las hojas, si existen elementos solapados prevalecerá el último elemento que haya sido dibujado.

Una de las principales características de implementación de la plataforma Android, es la posibilidad de declarar la arquitectura del *layout* a través de un archivo XML, el cual posee una estructura y codificación simples que permite ser leída por humanos. Los elementos de estos archivos pueden ser de tipo *ViewGroup* o *View*. En la figura 2.5 se puede observar un *layout* lineal (*ViewGroup*) con orientación horizontal que contiene una imagen (*View*) y un elemento de texto (*View*). Este podría ser más complejo y contener a su vez otro tipo de elemento *ViewGroup* si así lo requiere la actividad.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:background="#ffffff">

      <ImageView android:id="@+id/CreatorPhoto"
                 android:src="@drawable/icon"
                 android:layout_width="wrap_content"
                 android:layout_height="wrap_content"/>

      <TextView  android:id="@+id/CreatorName"
                 android:text="Creator Name"
                 android:layout_width="fill_parent"
                 android:layout_height="wrap_content"/>
</LinearLayout>
```

Figura 2.5. Archivo XML del Layout de una Actividad.

Cada tipo de elemento XML en el archivo del *layout* tiene su representación de clase Java. El elemento <TextView> tiene una clase Java llamada *TextView*.

Como se había mencionado anteriormente, existe una subclase de objetos *View* denominados *widgets* que sirven de interfaz para la interacción con el usuario. La plataforma Android posee un grupo de *widgets* como botones, casillas de selección y campos de texto, ya implementados y listos para ser agregados de manera rápida en la UI. También existen otros *widgets* más complejos como relojes y controles de zoom. Estos *widgets* pueden ser personalizados, combinados o simplemente creados desde cero.

Una vez establecido el diseño y componentes de una UI, es posible implementar acciones que respondan a interacciones con el usuario e iniciar *intents*. Para notificarle esto a la aplicación, la plataforma posee diversas clases y métodos que le facilitan la labor al desarrollador.

2.2.5. Filosofía de Diseño

A pesar que algunos de los dispositivos móviles que podemos encontrar en la actualidad ofrecen funciones similares a las de un ordenador de escritorio, se debe tener siempre en cuenta que poseen limitaciones computacionales, de almacenamiento y de energía. Por esta razón, Android ha insistido en recordarle a los desarrolladores sobre estas

limitaciones y ha propuesto una serie de recomendaciones para sobrellevarlas. Esta filosofía de diseño está basada en tres pilares fundamentales:

- Rapidez
- Respuesta
- Transparencia

En ocasiones se suele pensar que si una aplicación corre lo suficientemente rápido significa que es eficiente. Esto no podría ser más falso y repercute negativamente en el uso de la batería. Este último aspecto es de suma importancia para los usuarios, los cuales pueden conocer a través de herramientas de software cuál aplicación posee un alarmante uso de energía. Por esto se debe intentar optimizar las aplicaciones, *no hacer nada que no sea necesario* y *no almacenar en memoria si se puede evitar*[5].

La velocidad con la que una aplicación responde o no, es uno de los factores más importantes del comportamiento de una aplicación ya que está directamente relacionada con la experiencia inmediata del usuario. Aunque una aplicación sea eficiente, se puede dar el caso que el usuario sufra problemas al tratar de navegar a través de ella.

El sistema Android posee un escudo en el caso que una aplicación no responda con un mínimo nivel de presteza. Si un evento de entrada de usuario demora más de cinco segundos o si un receptor de broadcast no ha finalizado en ejecutarse transcurridos diez segundos[6], el sistema dispara un mensaje al usuario llamado *Application Not Responding*, o ANR. En la figura 2.6 se puede observar el típico mensaje ANR y como permite al usuario escoger entre forzar el cierre o esperar que la aplicación reaccione.

Adicionalmente, el usuario desea poder interactuar con el sistema y otras aplicaciones de una forma eficaz y transparente. Aunque una aplicación sea rápida y responda de manera correcta a las entradas de usuario, algunas decisiones de diseño pueden ocasionar problemas de interacción con otras aplicaciones, pérdida de información, bloqueos inesperados, entre otras cosas. Para evitar este tipo de problemas, es necesario entender el contexto en el cual se ejecutan las aplicaciones y como las tareas del sistema que pueden afectar a la aplicación.

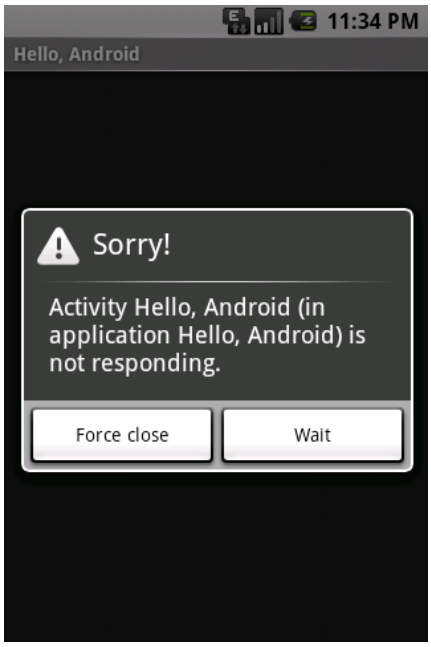

Figura 2.6. Mensaje ANR de una aplicación Android.

Como se mencionó en secciones anteriores el sistema Android está diseñado para tratar a las aplicaciones como un grupo de funciones encadenadas más que como un trozo fijo de código. Esto permite ver al sistema como una federación de estos componentes e integrarlos de una manera limpia y transparente con otras aplicaciones.

2.2.6 Android vs. iPhone

Aunque en la actualidad existen varios sistemas operativos móviles como Symbian, Microsoft Windows Mobile, Blackberry OS y OpenMoko, este apartado se enfocará en señalar las ventajas de Android respecto a su principal competidor, el iPhone de Apple. Ambos buscan llegar a un grupo similar de consumidores que buscan dispositivos de pantalla táctil, con fácil conexión a Internet, aplicaciones innovadoras y una UI atractiva.

Aunque Android mejoraba al iPhone OS en muchos niveles desde su aparición en el mercado, no ofrecía todas las funcionalidades y atractivo comercial del móvil de Apple. Sin embargo, Android ha crecido desde entonces ofreciendo todas las funcionalidades del iPhone en una extensa variedad de dispositivos y un mercado cada vez más grande.
A continuación se presentarán las principales características que Android ofrece a sus usuarios sobre el iPhone OS[27].

En primer lugar Android puede correr múltiples aplicaciones al mismo tiempo, independientemente si estas son aplicaciones nativas o si provienen del mercado de aplicaciones Android, mientras que iPhone OS ofrece un soporte de multitarea limitado para aplicaciones nativas. En su última versión, el iPhone OS 4, Apple intentó nivelar esta deficiencia otorgando a los desarrolladores acceso a una limitada variedad de APIs que pueden ser ejecutadas en el fondo. Esto ni se acerca a las bondades que Android ofrece a sus usuarios, los cuales pueden recibir notificaciones, escuchar música o almacenar datos GPS sin tener que mantener las aplicaciones abiertas.

Adicionalmente, Android mantiene más información disponible en sus múltiples y personalizables pantallas de inicio otorgando al usuario accesibilidad inmediata a sus aplicaciones sin tener que ejecutarlas. Por otra parte, los usuarios de iPhone tienen que localizar las aplicaciones en una lista y posteriormente ejecutarlas. Si por ejemplo se desea obtener información del clima, habrá que buscar la aplicación, abrirla y esperar que se ejecute. Mientras que en Android existen *widgets* para cualquier aplicación, haciéndolas disponibles directamente en la pantalla de inicio.

Otro aspecto importante es el mercado de aplicaciones de Android. Aunque es cierto que iPhone ofrece alrededor de 200.000 aplicaciones[12] en comparación con las poco más de 80.000 de Android[8], este último presenta un crecimiento más acelerado. Adicionalmente, las aplicaciones de iPhone son evaluadas antes de ser publicadas en el mercado, muchas siendo ignoradas por considerarse simples o parecidas a otras aplicaciones. El mercado de aplicaciones de Android es completamente abierto y regido por los consumidores, la mejor aplicación es la que tendrá más apoyo por los usuarios. Adicionalmente, Android no censura las aplicaciones lo cual ofrece un mercado ilimitado y libre.

Otra ventaja de Android son las notificaciones. El iPhone sólo permite notificaciones *pop-up* las cuales únicamente pueden ser manipuladas una por una y, por sus limitaciones de multitarea, las aplicaciones deben estar abiertas para poder mostrar estas notificaciones. No obstante, Android posee una barra de notificaciones desplegable con un listado de notificaciones pendientes. Adicionalmente, Android permite a los desarrolladores mostrarlas desde la pantalla de suspensión, algo que iPhone sólo ofrece para aplicaciones nativas.

Actualmente los *Smartphones* proporcionan conectividad constante permitiendo desempeñar la mayoría de las labores virtuales desde cualquier punto. Android ofrece la posibilidad de integrar esta conectividad de manera nativa. Por ejemplo, es posible sincronizar automáticamente la cuenta de correo electrónico de *Gmail*[37] con el dispositivo móvil o automáticamente colgar en *Flickr*[33] las fotos capturadas desde el teléfono. iPhone únicamente puede hacer esto a través de aplicaciones de terceros y no de una manera igualmente transparente.

Una de las principales de ventajas de Android respecto a iPhone es que, al ser una plataforma abierta, permite a diferentes manufacturadores ofrecer dispositivos que trabajen con Android. En la actualidad existen más de 60 dispositivos fabricados por 21 compañías diferentes[36]. Si comparamos esto con Apple, que únicamente ofrece un dispositivo para iPhone OS, se puede observar una importante diferencia en la diversidad de opciones a las que los consumidores pueden optar. Esta diversidad también se ve reflejada en el precio de los dispositivos, los cuales abarcan un rango más amplio y permiten a diferentes *bolsillos* optar por estos móviles. Adicionalmente, al no existir ningún convenio de exclusividad, todas las compañías proveedoras de servicios pueden dar soporte y ofrecer a sus clientes dispositivos móviles Android.

2.3. Entornos de Desarrollo

En esta sección se presentarán todos aquellos programas de software y entornos de desarrollo utilizados para la realización de la aplicación. En primer lugar se definirá el *Software Development Kit*, o SDK, de Android que contiene todas las librerías Java necesarias para el desarrollo de una amplia variedad de aplicaciones. También se explicará con cierto detalle el módulo *Android Development Tool* para Eclipse, el cual permite desarrollar aplicaciones Android en este ambiente facilitando el uso de algunas herramientas del SDK. Adicionalmente, se explicarán algunas de las APIs más importantes, tanto para la implementación de este proyecto en concreto, como para la mayoría de las aplicaciones Android. Otro aspecto a tratar en esta sección será la aplicación del emulador de dispositivos Android y su respectiva consola. Ambas herramientas permiten a los desarrolladores probar sus aplicaciones dentro de un contexto real sin la necesidad de disponer de un dispositivo móvil físico. También se

hablará brevemente sobre la versión J2ME de java y se establecerán sus diferencias más importantes respecto a la plataforma Android. Finalmente, se mencionarán algunos factores de importancia de la aplicación de software Wireshark para el desarrollo de este proyecto.

2.3.1 Android Software Development Kit

La plataforma Android incluye un conjunto de librerías base que otorgan al entorno de desarrollo la mayor parte de las funcionalidades disponibles en las librerías base de Java. A su vez, es posible añadir algunas librerías desarrolladas en J2SE y la mayoría de las desarrolladas en J2ME.

Dentro de este kit de desarrollo también se pueden encontrar útiles herramientas que le otorgan al programador flexibilidad al momento de desarrollar aplicaciones en esta plataforma.

Una de las principales herramientas es el emulador de dispositivos móviles Android. Este incorpora todas las características de un móvil de última generación excepto las funciones de teléfono. El emulador permite ajustar la velocidad de conexión, dimensiones de la pantalla, si esta es táctil o no, entre otras cosas. Las características del emulador se tratarán en mayor detalle en la sección 2.3.5.

También existen dos herramientas para la depuración y configuración de los datos del emulador. La primera de ellas es el *Dalvik Debug Monitor Service* (DDMS) y el *Android Debug Bridge* (ADB). El DDMS ofrece la posibilidad de depurar las aplicaciones e interactuar con diferentes instancias del emulador de una forma visual e interactiva, en especial a través del módulo de Android para Eclipse. El DDMS permite, entre otras cosas, importar y exportar archivos entre la instancia del emulador y el ordenador donde se esté desarrollando la aplicación. Por otra parte, el ADB permite abrir un terminal Linux del emulador con el cual es posible instalar y eliminar aplicaciones y modificar, crear y eliminar directorios y archivos. Sin embargo, a partir de la versión 1.5 del SDK, el emulador incluye por defecto un terminal Linux para realizar estas tareas, incorporado en las herramientas del dispositivo.

Como se mencionó anteriormente cada una de las aplicaciones Android se ejecutan en procesos diferentes. Aunque esto representa ventajas en la ejecución de tareas múltiples, complica la compartición de objetos entre diferentes aplicaciones de la plataforma. Para poder compartir información, los procesos se han de descomponer en objetos primitivos para poder ser entendidos por el sistema. Para poder realizar esta tarea, la plataforma ofrece el *Android Interface Description Language* (AIDL). Como su nombre lo dice, esta herramienta es un lenguaje IDL[69] utilizado para general el código que hará posible que dos procesos se comuniquen en un dispositivo Android.

El SDK también permite simular el almacenamiento de datos de una tarjeta SD. Para ello, se tiene una herramienta llamada *mksdcard*, con la cual se puede crear una imagen ISO de tamaño y nombre deseados. Esto permite al emulador, y a las aplicaciones que se ejecuten en el mismo, hacer uso de una memoria extraíble simulada.

2.3.2 Módulo ADT para Eclipse

Eclipse[26] es un proyecto de código abierto multiplataforma basado en un entorno de desarrollo integrado (IDE) que contiene, en esencia, un editor de código, un compilador, un depurador y un constructor de UIs, utilizados para desarrollar aplicaciones de cliente enriquecido.

La plataforma proporciona un módulo del Android Development Tools (ADT) para Eclipse el cual añade extensiones de gran alcance para desarrollar aplicaciones Android. Este módulo permite accesar a algunas de las herramientas mencionadas en la sección anterior desde el entorno de Eclipse, como por ejemplo, acceder a las capacidades del DDMS, realizar capturas de pantalla, establecer puntos de interrupción, entre otras. También proporciona al desarrollador un asistente para crear nuevos proyectos de una manera fácil y rápida.

Otra herramienta que proporciona este módulo es el editor de código Android que ayuda a escribir archivos XML al momento de definir los recursos que dictarán la arquitectura del layout y en la declaración del manifiesto Android. Además de simplificar y automatizar la construcción de las aplicaciones, permite exportar el proyecto en un archivo APK firmado, listo para ser distribuido a los usuarios de la aplicación.

2.3.3 Android APIs

La plataforma Android ofrece un grupo de paquetes con interfaces y clases Java que permiten a los desarrolladores implementar un amplio rango de funcionalidades de alto nivel de una manera sencilla, sin tener que lidiar con componentes primitivos del sistema. En esta sección se explicarán las APIs esenciales para la implementación de *Tales* y otras que, aunque no fueron utilizadas concretamente en esta aplicación, son de suma importancia para un mayor entendimiento de algunos servicios que ofrece la plataforma.

APIs multimedia

El paquete *android.media* representa la API especializada para implementar funciones multimedia en las aplicaciones. Esta API permite la reproducción de audio y video a partir de ficheros contenidos en los recursos de la aplicación o en modo *streaming* para ficheros online. Para ello, únicamente se debe crear una instancia de la clase android.media.MediaPlayer, establecer la fuente de los datos que se desean reproducir y realizar los pasos de inicialización y finalización del reproductor. También es posible configurar acciones para los estados de reproducción, por ejemplo, la aplicación podría notificarle al usuario que una canción terminó o cerrar automáticamente el visualizador una vez que un video haya finalizado. Esta API ofrece un amplio rango de funciones, desde correr archivos en hilos de ejecución[67] independientes hasta iniciar reproducciones tipo JET[43].

Otra de las funciones de esta API es la posibilidad de grabar audio y video a través del hardware de entrada del dispositivo. Esto se puede hacer de manera análoga a la reproducción mediante la clase android.media.MediaRecorder. Para ello, se debe crear una instancia de esta clase e indicar la fuente de entrada de audio o video, especificar el formato y codificación de salida, indicar el lugar donde se desean almacenar los datos y por último realizar los pasos de inicialización y finalización de la grabación. La API también ofrece la clase android.media.AudioRecorder, diseñada especialmente para grabar archivos de audio haciendo uso de los bytes de entrada al buffer. Aunque esta es una solución de menor nivel, en ocasiones ofrece capacidades que la otra clase no puede proporcionar como, por ejemplo, en caso que la codificación de entrada deseada no sea soportada por la clase anterior. Este caso particular se explicará en mayor detalle en el

capítulo 4. En la tabla 2.1 se puede observar el listado de formatos y de codificación de entrada y salida que Android acepta para sus archivos de audio y video.

Type	Format	Encoder	Decoder	File Type(s) Supported
Audio	AAC LC/LTP		X	3GPP (.3gp) and MPEG-4 (.mp4, .m4a). No support for raw AAC (.aac)
	HE-AACv1 (AAC+)		X	
	HE-AACv2 (enhanced AAC+)		X	
	AMR-NB	X	X	3GPP (.3gp)
	AMR-WB		X	3GPP (.3gp)
	MP3		X	MP3 (.mp3)
	MIDI		X	Type 0 and 1 (.mid, .xmf, .mxmf). Also RTTTL/RTX (.rtttl, .rtx), OTA (.ota), and iMelody (.imy)
	Ogg Vorbis		X	Ogg (.ogg)
	PCM/WAVE		X	WAVE (.wav)
Video	H.263	X	X	3GPP (.3gp) and MPEG-4 (.mp4)
	H.264 AVC		X	3GPP (.3gp) and MPEG-4 (.mp4)
	MPEG-4 SP		X	3GPP (.3gp)

Tabla 2.1. Formatos y codificación de entrada y salida para audio y video.

API OpenGL

Como se mencionó anteriormente, la plataforma Android ofrece un motor para la implementación de gráficos 3D basado en OpenGL. Las clases e interfaces de esta API están contenidas en el paquete android.opengl. Este paquete permite realizar un gran número de operaciones GL, como codificar y decodificar texturas ETC1[45]. También ofrece un depurador de aplicaciones OpenGL ES y la clase GLUtils para puentear estas operaciones con otras APIs de Android. Adicionalmente, existe una clase base para cada una de las tres versiones de OpenGL para sistemas embebidos.

Utilizar esta API en alto nivel es relativamente sencillo, primero es necesario enlazar la aplicación a un OpenGLContext para accesar a las funcionalidades de OpenGL y luego,

dentro del método *onDraw*, obtener un *handle*[66] del objeto GL y definir los métodos necesarios para reproducir la escena.

APIs para servicios Google

Tal como se mencionó en el capítulo anterior, Google pertenece al consorcio del Open Handset Alliance y lidera el proyecto de la plataforma Android. Por ello, la plataforma dispone de APIs específicas para la gestión algunos servicios de *Google* como lo son *Google Maps* y *GTalk*.

Google Maps ofrece un completo servicio de mapas, con diferentes tipos de vistas, establecimiento de rutas personalizadas, localización de comercios y otras funciones típicas de este tipo de servicios. Para ello, es necesario extender al SDK las bibliotecas externas de *Google Maps* que permiten descargar, dibujar y capturar las *tejas* de los mapas, así como proporcionar opciones y controles de visualización. Las clases principales de la API son *MapView* y *MapActivity*. Esta última se encarga de de gestionar el funcionamiento de *MapView*, la cual ofrece todo tipo de interacción manual y visual con el usuario. Como esta API ofrece acceso a los datos de Google Maps, es necesario que el desarrollador esté registrado al servicio, acepte los términos de uso y proporcione una *huella* MD5[1] del certificado que utilizará para firmar sus aplicaciones.

Por otra parte, el servicio de GTalk permite a sus usuarios intercambiar mensajes de texto y establecer vídeo-llamadas. Para ello Google desarrolló un protocolo propio basado en el estándar XMPP[77]. Esto representa una gran ventaja para los usuarios ya que pueden gozar de estos servicios sin acarrear con los costes de mensajería y llamadas telefónicas tradicionales. Adicionalmente, la comunicación está gestionada de manera que los mensajes se transmiten con una velocidad mayor a la de los mensajes SMS gracias a una única conexión persistente que se encarga de todo el tráfico GTalk en el dispositivo.

API de geolocalización

Android también permite incorporar funciones de geolocalización en las aplicaciones. La plataforma ofrece el paquete android.location el cual posee interfaces y clases para este fin. Con esta API es posible accesar a los servicios del sistema de localización e indicar

los proveedores de GPS. Si el dispositivo móvil no es apto para recibir e interpretar este tipo de información, es posible obtener los datos de localización a través de algoritmos de triangulación de las radio-bases de los proveedores de red. Adicionalmente, esta API ofrece clases para el manejo de datos de posicionamiento e información sobre la dirección donde se está localizado. Una de las clases fundamentales contenida en esta API es el *LocationManager*, el cual, entre otras cosas, permite realizar acciones de manera automática al encontrarnos en una ubicación determinada. Un ejemplo de esto podría ser una guía de ocio que notifique al usuario que existe un lugar de interés turístico cerca de su ubicación actual.

Estas funciones pueden ser complementadas con la extensión de Google Maps. Esto permite crear un número importante de nuevas aplicaciones. Continuando el ejemplo de la guía de ocio, la aplicación podría mostrar un mapa indicando la ubicación del usuario y marcar la mejor ruta para llegar al lugar turístico del cual fue notificado.

API de hardware

Existe una API especialmente diseñada para dar soporte al hardware presente comúnmente en todos los dispositivos móviles Android. Para ello, la plataforma cuenta con el paquete android.hardware la cual cuenta con una serie de clases para la fácil implementación de funciones de hardware en las aplicaciones. La API cuenta con tres clases diferentes para gestionar las funciones de la cámara, desde la captura de fotos y video, hasta la elección de parámetros de imagen, orientación y codificación. Adicionalmente, se cuenta con la clase *GeomagneticField* la cual permite estimar el campo magnético de un punto determinado de la Tierra y calcular la declinación magnética desde el norte real.

Las últimas generaciones de *Smartphones* también cuentan, por lo general, con una serie de sensores embebidos. Este paquete ofrece tres clases que permiten capturar y manejar los datos que estos sensores proporcionan. También se pueden indicar etiquetas de tiempo y activar eventos de manera automática según los diferentes estados del acelerómetro o el sensor de inclinación.

APIs HTTP

La plataforma Android consta con un gran número de APIs para implementar cualquier tipo de funcionalidades HTTP en las aplicaciones. Estas APIs corresponden a los paquetes Java org.apache.http, los cuales están basados en el proyecto de código abierto de servidores HTTP de la Apache Software Foundation[11]. El objetivo de este proyecto es brindar servidores seguros, eficientes y extensibles que provean servicios HTTP dentro de los estándares actuales de esta tecnología. Desde 1996, se han posicionado como los servidores web más populares en Internet[10].

Este conjunto de paquetes son de suma importancia para aplicaciones que deseen interactuar con servicios web o comunicarse con APIs externas a través de Internet. A través de estas APIs es posible crear clientes HTTP y proporcionarles credenciales para autenticarlos con el servidor. También es posible implementar las de peticiones HTTP más comunes, al igual que peticiones POST multiparte basadas en MIME[71] para la transferencia de archivos. Adicionalmente, las APIs poseen clases especializadas para manejar las respuestas del servidor e iniciar acciones a partir de las mismas. A su vez, es posible monitorear y ser notificado en el caso de alguna incidencia en la conexión.

Todas estas tareas permiten ampliar considerablemente el alcance de las aplicaciones ya que permite extender sus funcionalidades con ayuda de procesos remotos, que de otra manera serían imposibles o muy costosos de implementar en los dispositivos móviles. Esta tecnología y sus implicaciones se presentarán en mayor detalle en secciones subsiguientes, al igual que las tareas para su implementación, que se explicarán en el capítulo 4.

API JSON

JavaScript Object Notation, o JSON[44], es un formato de notación ligero para el intercambio de datos. Este formato es utilizado en ocasiones como una solución diferente a XML por la simplicidad de su sintaxis y por la facilidad de implementar un analizador semántico en Java.

Los mensajes en este formato están compuestos únicamente por dos clases de elementos, *objetos* y *arreglos*. Los *objetos* están compuestos por una pareja de la etiqueta o nombre y su valor correspondiente. Por otra parte, los *arreglos* están compuestos por una serie de elementos que pueden estar formados por uno o más *objetos* JSON. Los documentos JSON están estructurados de manera jerárquica, fácil de entender y que puede ser leída claramente por humanos.

Android posee una API especial para JSON, que permite a los desarrolladores crear e interpretar este tipo de mensajes de una manera transparente e intuitiva. Estas clases permiten moverse con gran facilidad dentro de los arreglos y extraer los valores de cualquier objeto.

2.3.4 Emulador Android

Como se mencionó anteriormente la plataforma ofrece un emulador basado en el emulador de código abierto QEMU[55] con el cual los desarrolladores pueden probar sus aplicaciones tal como si se tratara de un dispositivo real. Este emulador ejecuta una imagen completa del sistema, la cual contiene el código de la máquina ARM[63] para el Linux kernel de Android, librerías nativas, la VM Dalvik, un amplio número de paquetes de la plataforma y aplicaciones nativas.

Además de las funciones típicas de emuladores QEMU, el emulador de Android ofrece las principales características de cualquier dispositivo móvil, como:

- Un CPU ARMv5 y la MMU[48] correspondiente
- Una pantalla LCD de 16-bits
- Un teclado Qwerty y botones de navegación
- Un chip para entrada y salida de audio
- Particiones de memoria Flash
- Un módem GSM, incluyendo una tarjeta SIM simulada

Para iniciar el emulador es necesario crear una o más configuraciones del Dispositivo Virtual Android, o AVD. En cada una de estas configuraciones se debe indicar la versión de la plataforma, las opciones de hardware y los parámetros de resolución y tamaño de la

pantalla. Luego al momento de ejecutar el emulador únicamente se debe seleccionar la configuración deseada de la AVD. No obstante, si se cuenta con el módulo para Eclipse es posible iniciar una aplicación sin especificar un AVD. En estos casos, la plataforma Android escogerá automáticamente la configuración de AVD más apropiada según la versión del SDK.

Cada AVD trabaja como un sistema independiente con su propio almacenamiento de información. También es posible crear imágenes de disco en el ordenador de desarrollo y asociarlas a una AVD con el fin de simular el uso de memoria extraíble. Esto permite a la aplicación leer y escribir datos durante la ejecución tal y como lo haría un usuario con la tarjeta SD en el dispositivo. Es importante notar que el emulador almacena por defecto en el directorio de la AVD la información de usuario, los datos de la tarjeta SD y la caché.

El emulador también permite configurar y probar modelos de red en las aplicaciones. Cada instancia del emulador corre detrás de un enrutador o firewall virtual, lo cual lo aísla de las configuraciones e interfaces de la máquina de desarrollo. Por ello, los dispositivos virtuales no pueden *ver* a la máquina de desarrollo u otras instancias del emulador. Estos sólo pueden *ver* que están conectados a un enrutador o firewall a través de una conexión Ethernet.

Las instancias del emulador se ejecutan como una aplicación cualquiera de la máquina de desarrollo y están sujetas a las mismas limitaciones de red. No obstante, es posible establecer comunicación con una instancia del emulador mediante redirecciones de red al enrutador virtual. Para ello, es necesario conectarse a un puerto *guest* del enrutador, mientras este direcciona el tráfico de entrada y salida al puerto *host* del dispositivo virtual.

La manera más común para realizar esta redirección es a través de la consola del emulador. A continuación se explicarán esta y otras funciones de esta útil herramienta.

La consola del emulador

Cada instancia del emulador posee una consola mediante la cual es posible simular y configurar una serie de eventos de manera dinámica. Mediante esta herramienta es posible configurar redirecciones de red, emular datos de geolocalización, iniciar eventos

de hardware, controlar el estatus de la batería, emular retardos en la red, incluso realizar llamadas y enviar SMS.

Para accesar a la consola del emulador primero es necesario iniciar una conexión *Telnet*[60] al puerto de la consola. Cada consola ocupa dos puertos adyacentes comenzando por los puertos 5554 y 5555 para la primera instancia del emulador. El primero de estos corresponde al puerto de la consola y el segundo al puerto del ADB. El comando para accesar a la consola de una instancia del emulador podría ser el siguiente:

telnet localhost 5554

Los puertos que se pueden utilizar se encuentran en el rango del puerto 5554 hasta el puerto, ambos inclusive, lo cual permite mantener 16 instancias diferentes del emulador, cada uno con su respectiva consola.

Como se mencionó anteriormente es posible realizar redirecciones de puerto a través de la consola una vez se haya iniciado una instancia del emulador. Para ello, es necesario hacer uno del comando redir de la siguiente forma:

redir add <protocolo> <puerto-host> <puerto-guest>

Los protocolos pueden ser TCP[24] o UDP[53], mientras que los puertos *host* y *guest* corresponden a los puertos de la máquina de desarrollo y del emulador respectivamente. Adicionalmente es posible eliminar redirecciones mediante el subcomando del, o mostrar un listado de las redirecciones de una instancia con el subcomando list.

Emulación de geo-localización

La consola ofrece comando geo que permiten establecer una posición ficticia al emulador como si estuviese conectado a un proveedor GPS. Este comando soporta dos tipos de formato, fix y NMEA[48]. Estos datos de localización ficticia pueden ser usados por aplicaciones una vez se inicie la instancia del emulador y probar funciones de este tipo. Cabe destacar que es posible modificar la localización tantas veces sea desee haciendo uso de este comando.

Emulación de eventos de hardware

Como se mencionó anteriormente, es posible simular eventos de hardware a través de la consola del emulador, desde enviar eventos a kernel de Android hasta la pulsación de teclas. El comando event es el encargado de estas funciones. En la tabla 2.2 se muestra la sintaxis y subcomandos de event.

Subcomando	Descripción
send <type>:<code>:<value>[…]	Envía uno o más eventos al kernel de Android
types	Muestra el listado de todos los aliases <type> soportados por los subcomandos de event
codes <type>	Muestra el listado de todos los aliases <codes> soportados por los subcomandos de event de un <type> específico
event text <mensaje>	Simula eventos de teclado para ser enviados como un mensaje

Tabla 2.2. Subcomandos del comando de consola event.

Emulación de latencia y velocidad de transmisión

La consola también permite emular la latencia en las conexiones de red para crear condiciones de comunicación más reales. Es posible establecer un rango de valores de latencia durante la inicialización del emulador. Para ello, se debe hacer uso del comando emulator -netdelay <rango de latencia>. Si en lugar de esto se desea establecer dinámicamente la latencia durante la ejecución de la aplicación, se debe utilizar el comando de consola network delay <rango de latencia>. Para ambos casos se puede indicar un rango numérico determinado o especificar una tecnología de conexión con un rango predefinido. Actualmente la consola posee emulación para GPRS, EDGE/EGPRS y UMTS/3G.

También es posible simular diferentes tasas de transmisión en la red. Al igual que en el caso de la latencia, también es posible fijar un rango de tasa de transmisión durante la inicialización o bien modificarla manualmente a través de la consola mientras la aplicación es ejecutada. Las tecnologías de red soportadas para este caso son: GSM/CSD, HSCSD, GPRS, EDGE/EGPRS, UMTS/3G y HSDPA. Las líneas de comando para realizar estas acciones son:

emulator -netspeed <velocidad de transmisión>

y

network -speed <velocidad de transmisión>

respectivamente.

Emulación de telefonía y mensajería

El emulador también permite simular eventos de llamadas gracias a su módem GSM virtual y tratarlos de la misma manera que lo haría un dispositivo móvil real. No obstante, como era de esperarse, estas llamadas no son transmitidas por las redes de telefonía ni soportan audio. Las llamadas son únicamente simuladas a nivel de aplicación. Una vez se esté conectado a la consola, el comando call permite la emulación de recibir, emitir, contestar y cancelar llamadas, o incluso ponerlas en espera. También es posible modificar el estado de la conexión GPRS y listar el historial de llamadas. A continuación se muestra la sintaxis del comando call y sus subcomandos:

gsm <call|accept|busy|cancel|data|hold|list|voice|status>

A través de la consola también se pueden generar mensajes SMS y enviarlos a una instancia del emulador. Para ello, se debe hacer uso de la siguiente línea de comando:

sms send <número telefónico> <mensaje de texto>

El número telefónico puede ser una cadena arbitraria de números ya que, igual que las llamadas, no posee repercusión en la red telefónica.

2.3.5 Android/J2ME

Es importante tener en cuenta las diferencias entre Android y J2ME. J2ME es un lenguaje de programación de Sun Microsystems que incorpora un grupo de librerías especialmente diseñadas para trabajar en dispositivos móviles. Por otra parte, Android es una plataforma completa que, como se ha mencionado anteriormente, incluye un sistema operativo con un entorno de desarrollo basado en el lenguaje de programación Java. Otra diferencia fundamental es que la plataforma Android usa la máquina virtual Dalvik, mientras que J2ME utiliza la máquina virtual de Sun. La máquina virtual Dlavik no puede interpretar ficheros *.class.* Sin embargo, cualquier librería desarrollada en J2ME puede ser llevada a Android por medio de la herramienta *dex* del entorno de desarrollo.

También se pueden establecer comparaciones en cuanto al desarrollo de las aplicaciones. La plataforma Android ofrece un módulo para el ambiente de Eclipse, el cual trabaja de manera análoga al Netbeans de J2ME. Aunque existen librerías que se encuentran en ambas plataformas, tienen APIs completamente diferentes. Adicionalmente, las aplicaciones que se desarrollan para Android sólo están disponibles para este sistema. Sin embargo, las aplicaciones de J2ME sólo necesitan una máquina virtual que pueda interpretar sus ficheros en Android.

2.3.6 API del MTG

Como se mencionó anteriormente, parte de los objetivos de este proyecto se basan en la integración de tecnología propia del Music Technology Group en la aplicación Tales. Entre estas tecnologías se destacan una API con la cual la aplicación realiza peticiones a los servidores del MTG que albergan los servicios web y la base de datos.

Esta API está basada en una arquitectura tipo *Representational State Transfer* o REST, la cual realiza todas sus peticiones y respuestas por medio de peticiones HTTP. Sin embargo, vale la pena destacar que esta arquitectura puede estar basada en otros protocolos de la capa de aplicación.

La arquitectura REST se basa en clientes y servidores. Los primeros inician peticiones a los servidores quienes la procesan y retornan una respuesta coherente. Estas peticiones y respuestas están basadas en la transferencia de la representación de recursos, los cuales pueden ser cualquier tipo de concepto que pueda ser identificado por una dirección. Por otra parte, la representación de estos recursos pueden ser documentos que capturen el estado actual o deseado de estos últimos.

Un cliente puede estar en transición entre estados de una aplicación o en descanso, o *at rest*. Este último significa que el cliente puede comunicarse con su usuario pero no genera carga y no consume capacidad de almacenamiento en los servidores o en la red. El cliente envía peticiones al servidor cuando desea realizar una transición a un nuevo estado. La representación de los estados de una aplicación contiene enlaces que pueden ser utilizados por el cliente al iniciar un nuevo estado de transición.

En general, para que un sistema sea considerado como RESTful debe cumplir una serie de condiciones globales, mientras que otros aspectos más específicos se dejan libres para el diseño de la implementación.

En primer lugar, los clientes y los servidores se encuentran separados por una interfaz uniforme, donde el cliente no se preocupa por el almacenamiento de datos dejando esta tarea a los servidores. Por otra parte, los servidores no se preocupan por la interfaz o estado de los clientes mejorando la simplicidad y escalabilidad del sistema. Estos dos conceptos permiten que tanto los clientes como los servidores sean implementados de manera separada siempre y cuando se respeten la uniformidad de la interfaz.

Como se mencionó en el párrafo anterior, no son los servidores sino los clientes quienes se encargan de almacenar información sobre su estado. Sin embargo los servidores pueden ser *stateful*, o con estado, haciendo que su estado pueda ser identificado como un recurso mediante una URL. Esto facilita el monitoreo del servidor haciéndolo más visible. Un sistema RESTful puede tener también servidores intermedios transparentes para el cliente ayudando a una mayor escalabilidad y mayor balance de la carga mediante caches compartidas.

Esta API utiliza las típicas peticiones y respuestas del protocolo HTTP para la comunicación cliente-servidor y serán explicadas posteriormente en el apartado 2.4.1 de este documento. De esta misma manera las direcciones de los recursos utilizados por la aplicación serán indicadas en el capítulo de implementación.

2.3.7 Wireshark

Aunque la aplicación de software libre Wireshark no puede ser considerada como un entorno de desarrollo, fue una herramienta frecuentemente usada durante el desarrollo de este proyecto.

Wireshark es un popular analizador de protocolos que permite capturar todo el tráfico que pasa a través de una red y observar los campos que componen cada uno de sus paquetes. Esta aplicación es de suma importancia al momento de analizar y solucionar problemas en redes de comunicación. Su versión 1.4.1 es compatible con más de 480 protocolos y puede leer archivos de captura de más de 20 productos. Adicionalmente, posee una interfaz gráfica flexible que permite establecer filtros de visualización para restringir cómodamente los mensajes dentro de la comunicación y facilitar la terea de análisis.

En el capítulo 4 se mostrarán algunas capturas que permitirán al lector comprender más a fondo los detalles de implementación de la aplicación *Tales*.

2.4 Protocolos y Tecnologías

Aunque la aplicación Tales se ejecuta dentro del contexto de aplicaciones independientes, realiza la mayoría de sus funciones con la ayuda de un servidor remoto. Este servidor gestiona la base de datos que contiene toda la información que se le presentará al usuario a través del dispositivo móvil, desde texto e imágenes hasta archivos de audio. Adicionalmente, este servidor se encarga de ejecutar tareas que representarían un coste de procesamiento muy alto para un dispositivo móvil. En el servidor también existe una API, ajena a las de la plataforma Android, que se encarga de gestionar todo el intercambio de información mencionado. Por estas razones los protocolos de comunicación son de

suma importancia para este proyecto. Algunos conceptos básicos del protocolo HTTP serán tratados en esta sección.

Por otra parte, existen algunas tecnologías o estándares que dictan la manera en que se debe presentar la información para poder interpretarla de manera correcta y, así, utilizarla en las diferentes funcionalidades de la aplicación. Otros aspectos que se tratarán, son los formatos audio utilizados lo que permitirá entender algunas soluciones clave en la implementación de este proyecto.

2.4.1 HTTP

El *Hypertext Transport Protocol* o HTTP (en catellano Protocolo de Transporte de Hipertexto), es un protocolo desarrollado conjuntamente por la *Internet Engineeing Task Force* (IETF) y la *World Wide Web Consortium* (W3C) con la finalidad de establecer un estándar para la creación de redes en sistemas de información distribuidos, colaborativos y de hipermedia[31]. Actualmente HTTP es uno de los protocolos más importantes de la pila de protocolos de Internet y es considerado como la base para la comunicación de datos en la *World Wide Web*. La versión mayormente distribuida es la 1.1, la cual se encuentra definida en el RFC 2616, creado en 1999.

HTTP trabaja como un protocolo de petición-respuesta en el modelo de cliente-servidor. Dentro de este modelo, un cliente o *user agent*, que puede ser un buscador o una aplicación móvil, envía una petición HTTP al servidor. Este interpreta esta petición y realiza la acción que el cliente solicitó, ya sea almacenar contenido u otorgar algún recurso. Estos recursos son identificados y localizados en la red a partir de sus Uniform Resource Identifiers, o URI[17].

En las definiciones de HTTP se asume cierto nivel de confianza en la comunicación de la capa de transporte, por lo cual se ubica generalmente sobre TCP[74]. No obstante, se ha demostrado que HTTP puede trabajar correctamente en algunas aplicaciones sobre protocolos de transporte no fiables como UDP[53].

Para poder realizar las transacciones de petición-respuesta mencionadas anteriormente en esta sección, se debe iniciar una conexión TCP a un puerto del host, generalmente el

puerto 80 o *puerto HTTP*. Una vez establecida esta conexión el servidor HTTP que escucha ese puerto espera que el *user agent* envíe una petición. Posteriormente el servidor envía la respuesta correspondiente a través del puerto. El formato de estas peticiones y respuestas se explicará a continuación.

Mensajes de petición

Los mensajes de petición por parte del *user agent*, contienen en primer lugar la línea de petición donde se debe indica la acción o método que el servidor debe realizar. Entre los métodos aceptados por este protocolo tenemos:

- GET: solicita la representación de un recurso específico.
- HEAD: solicita una respuesta idéntica a la de una solicitud GET, pero sin contener el cuerpo de la respuesta. Es generalmente usado para obtener la *metadata* contenida en la cabecera de la respuesta, sin necesidad que el servidor envíe el contenido del recurso.
- POST: presenta datos para ser procesados por el servidor sobre el recurso indicado, ya sea la creación de uno nuevo o la actualización de un recurso existente.
- PUT: permite cargar una representación del recurso especificado.
- DELETE: borra el recurso identificado por la URI.
- TRACE: envía de vuelta la petición solicitada. Esto permite al cliente ver si servidores intermedios han realizado cambios en el recurso.
- CONNECT: es utilizado en proxys para cambiar dinámicamente la conexión a modo túnel para facilitar comunicación encriptada.
- PATCH: es utilizado para modificar parcialmente algún recurso.
- OPTIONS: solicita al servidor los métodos que este soporta para un recurso determinado.

Después del método, la línea de petición indica el identificador o URI del recurso. Este permite al servidor ubicar dicho recurso y realizar las acciones pertinentes sobre el mismo. Por último, la línea de petición indica qué versión del protocolo se encuentra en uso. La versión actual es la 1.1.

Posteriormente a la línea de petición se encuentra la cabecera. Esta permite al cliente enviar al servidor información adicional acerca de la petición o acerca del cliente mismo. La cabecera está formada por una serie de campos formados por su nombre y valor, separados por dos puntos ":". Estos campos permiten indicar los tipos de media, codificación o lenguaje de respuesta permitidos por el cliente. También permiten al usuario enviar credenciales para autenticarse frente al servidor, identificar el host del recurso solicitado, indicar el número máximo de saltos de la petición, entre otras cosas. Después de todos los campos hay una línea en blanco que indica el fin de la cabecera y el comienzo del cuerpo del mensaje de petición.

Mensajes de respuesta

Después de recibir e interpretar la petición de *user agent*, el servidor HTTP envía un mensaje de respuesta. Este, de manera análoga a la petición, está formado por la línea de estatus del mensaje, los campos de cabecera y el cuerpo del mensaje.
La línea de estatus está formada primeramente por la versión del protocolo en uso, seguida por el código del estatus. Estos últimos están representados por tres dígitos, de forma que el primero identifica su clase:

- 1xx: son de carácter informativo e indican que la petición fue recibida y será procesada.
- 2xx: indican una petición exitosa. La solicitud fue recibida, entendida y aceptada.
- 3xx: indican redirección, es decir, que se necesitan realizar más acciones para completar la solicitud.
- 4xx: indican una falla del cliente. Ocurren cuando existe un error de sintaxis o la petición solicitada no es válida.
- 5xx: indican errores por parte del servidor al momento de realizar una acción aparentemente válida.

Al igual que en los mensajes de solicitud, los mensajes de respuesta también posee una cabecera. Esta posee campos en los cuales el servidor puede enviar información adicional al cliente que no puede ser indicada con el código de estatus. Los campos pueden contener, entre otras cosas, información acerca del servidor, fecha, autenticación, control de caché, lenguaje y codificación del mensaje.

En versiones anteriores del protocolo, las conexiones se cerraban una vez enviados los mensajes de solicitud y respuesta. En la versión actual, es posible especificar una conexión persistente mediante el campo *Connection* de la cabecera del mensaje de solicitud. Esto permite reducir el retraso que representan las negociaciones necesarias para iniciar una conexión TCP para cada petición.

HTTP es un protocolo sin estado, con lo cual no es necesario que el servidor almacene información o el estatus de cada uno de los usuarios durante las peticiones. Aunque esto puede ser considerado como una desventaja para algunas aplicaciones, puede ser solucionado mediante el uso de *cookies*[68].

Otro aspecto importante de este protocolo es la posibilidad de implementar una comunicación cliente-servidor segura. Existe una variante de HTTP llamada *Secure HTTP*[56] especificado en el RFC 2660. Las dos maneras para establecer el esquema de seguridad son el esquema de URI de https y el campo *Upgrade* de la cabecera de HTTP 1.1. El primero es sintácticamente igual al HTTP convencional, pero solicita que se añada una capa de SSL/TLS[25] para garantizar la seguridad. SSL es específicamente útil para comunicaciones HTTP ya que puede otorgar cierta seguridad aunque sólo el servidor se autentifique. En el caso del uso del campo *Upgrade*, se envía inicialmente el mensaje en texto plano y posteriormente se solicita que la comunicación se pase a TLS. Tanto el cliente como el servidor pueden solicitarlo, aunque por lo general es el cliente quien envía el mensaje en texto plano y el servidor solicita el cambio a TLS.

2.4.2 WAVE

WAVE o WAV, apócope de *Waveform Audio Format*[32], es un formato de archivos de audio desarrollado conjuntamente por Microsoft e IBM como estándar para almacenar flujos de bits en ordenadores personales. WAVE está basado en RIFF[72], o *Resource Interchange File Format*, también creado por Microsoft e IBM como un formato genérico para almacenar información en bloques, o *chunks*, en ordenadores compatibles con IBM. RIFF, a su vez, está basado en el formato IFF[42], o *Interchange File Format*, utilizado frecuentemente en ordenadores Apple Macintosh y Amiga. RIFF fue expuesto en 1991 como el formato estándar para archivos multimedia de Windows 3.1.

Debido a la popularidad de Windows y la gran cantidad de programas de software desarrollados para la plataforma, WAVE es uno de los formatos de archivos de audio digital más usados en la actualidad. Se podría decir que casi cualquier programa moderno que almacene archivos de audio digital soporta este formato. No obstante, el formato WAVE es compatible con otros sistemas operativos como Linux y Apple Macintosh.

Por otra parte, los archivos WAVE no comprimidos son relativamente grandes. Esto ha hecho que WAVE pierda popularidad dentro del contexto tecnológico actual, que premia más una rápida transferencia que una alta calidad de sonido. No obstante, WAVE se continúa usando para almacenar los archivos originales donde el espacio en el disco no presenta mayor inconveniente o en aplicaciones para la edición de audio. En ocasiones el uso ubicuo de WAVE está relacionado con su familiaridad y su simple estructura por estar basado en RIFF. Los archivos WAVE no comprimidos también son utilizados en estaciones de radiodifusión modernas donde se ha adoptado un sistema digital.

Existen algunas limitaciones intrínsecas en el formato, como el tamaño máximo del archivo el cual no puede exceder los 4GB, lo cual representa aproximadamente 6,8 horas de audio en calidad de CD. Esta limitación se debe a un campo de 32 bits en la cabecera que indica el tamaño del archivo. Adicionalmente existen algunas inconsistencias en el formato, como por ejemplo, variaciones en la representación de los bits de muestreo según su valor e información redundante en diferentes bloques de datos. Otro factor negativo a destacar, es la incapacidad de proporcionar campos de información de título, artista, álbum, etc., utilizados frecuentemente en el intercambio de música.

Los archivos WAVE están usualmente conformados por un archivo RIFF y un bloque WAVE compuesto a su vez por dos sub-bloques, *fmt* que especifica el formato de los datos y *data* que contiene los bytes de datos[75]. En la figura 2.6 se muestran todos los campos de esta forma de representación de archivos WAVE.

A continuación se describirán cada uno de los campos del archivo WAVE de la figura 2.7:

- ChunkID: contiene las letras “RIFF” en formato ASCII.
- ChunkSize: indica el tamaño en bytes del archivo WAVE a partir de este campo.

- Format: contiene las letras “WAVE” en formato ASCII.
- SubChunk1ID: contiene las letras “fmt” en formato ASCII.
- SubChunk1Size: 16 para PCM. Indica el tamaño en bytes de este sub-bloque a partir de este campo.
- AudioFormat: 1 para PCM. Otros valores indican cierto tipo de compresión.
- NumChannels: indica el número de canales de audio.
- SampleRate: indica la frecuencia de muestreo. Estos deben ser compatibles con las frecuencias establecidas en el estándar.
- ByteRate: indica el número de bytes por cada segundo de audio.
- BlockAlign: indica el número de bytes por muestra, incluyendo todos los canales.
- BitsPerSample: indica el número de bits por muestra.
- Subchunk2ID: contiene la palabra “data” en ASCII.
- SubChunk2Size: indica el número de bytes de datos crudos.
- Data: representan los bytes de datos crudos, es decir, donde se encuentra contenida la información del sonido.

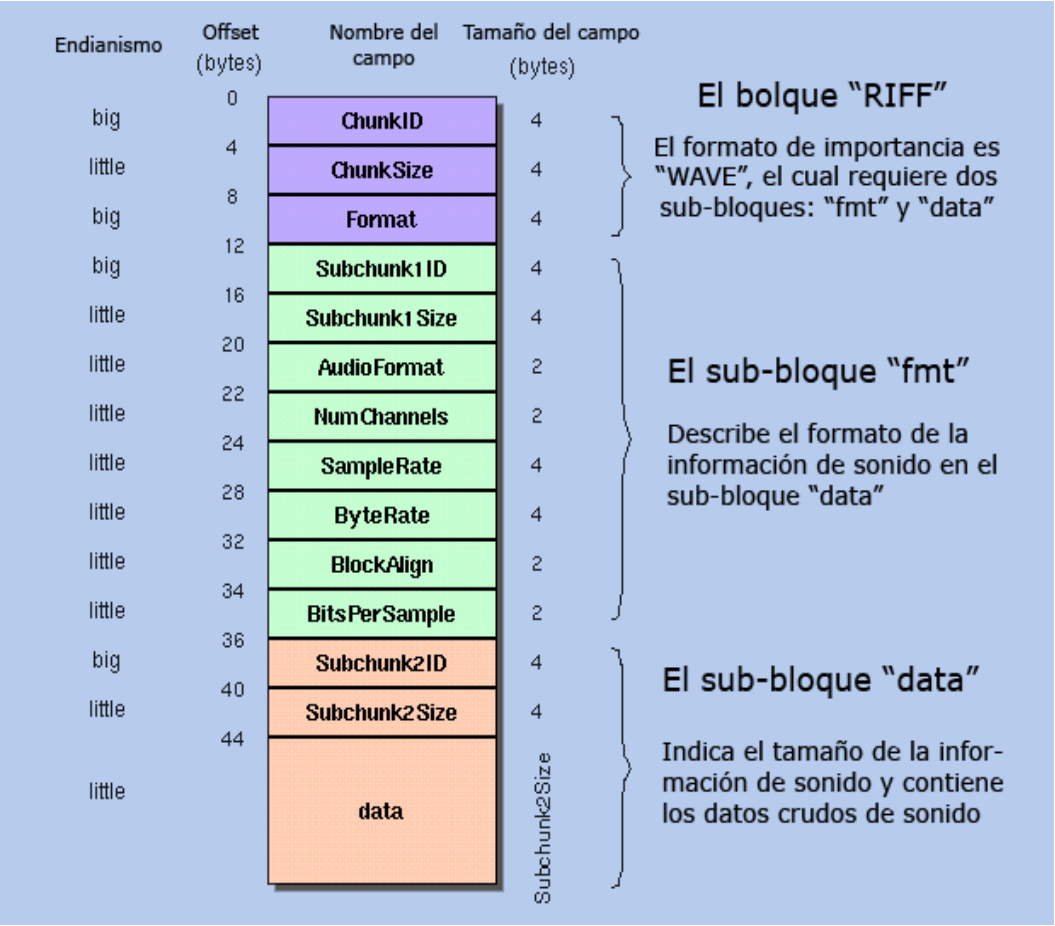

Figura 2.7. Bloques y campos de un típico archivo WAVE.

Capítulo 3

Actividades de la Aplicación

Tal y como se mencionó en capítulos anteriores, uno de los elementos fundamentales de cualquier aplicación bajo la plataforma Android son las *Activities* o Actividades. Estás representan los aspectos de la aplicación directamente vinculados a la GUI, donde cada una representan una pantalla. De esta manera, este capítulo busca explicar las funcionalidades de la aplicación desde el punto de vista del usuario. Se presentarán, pantalla por pantalla, el flujo de navegación dentro de la aplicación, qué hace cada botón y cómo se presenta la información. Este capítulo puede ser visto como un manual de usuario para ayudar a los lectores entender concretamente para qué sirve la aplicación y cómo interactuar con ella.

En la figura 3.1 se muestra un simple diagrama de flujo que describe la navegación a través de la aplicación. La primera pantalla corresponde al acceso o registro del usuario desde donde el usuario podrá comenzar a interactuar con la aplicación. Luego se muestra al usuario la vista con la descripción de la historia donde podrá participar. Aquí se muestran básicamente el creador y los personajes que participan en la historia. Desde esta pantalla el usuario podrá dirigirse a la vista con las partes que componen actualmente a la historia. Esta se basa en un listado con los segmentos de audio que los diferentes usuarios han grabado en esta historia. Por último se tiene la pantalla de participación en la historia donde el usuario podrá agregar nuevas partes a la misma y

subirlas al servidor. Cada una de estas pantallas se explicarán a continuación a lo largo de este capítulo.

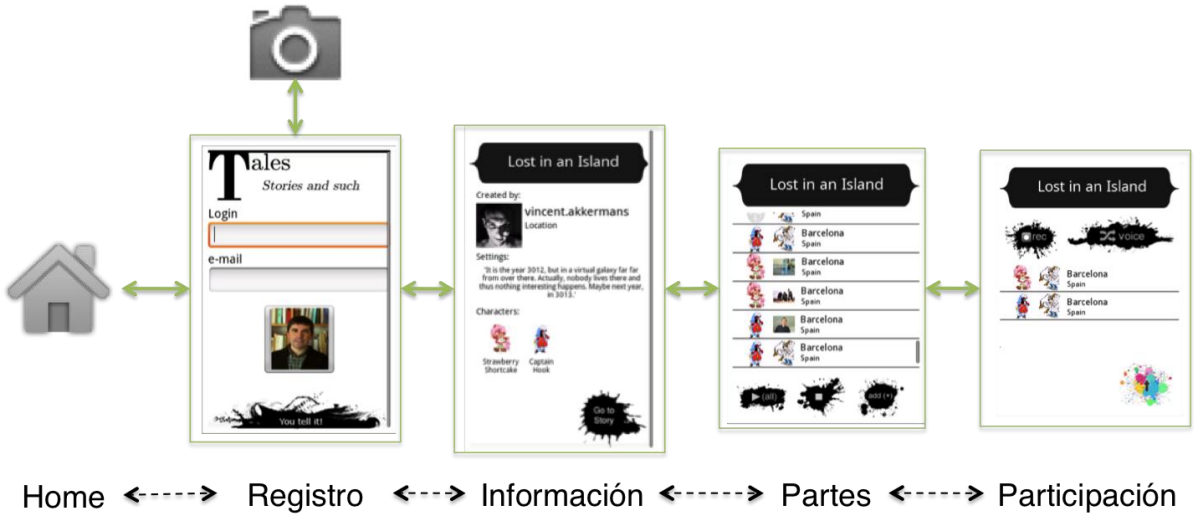

Figura 3.1. Diagrama de navegación de Tales.

3.1 Acceso y Registro de Usuario

La primera pantalla que se le presenta al usuario es la pantalla de *Acceso y Registro de Usuarios*. Una captura de esta se puede observar en la figura 3.2. En esta pantalla los usarios nuevos podrán registrarse y accesar a la aplicación. Para ello deberá ingresar una cuenta válida de correo electrónico y nombre de usuario cliqueando los respectivos campos de texto e introduciéndo los datos pertinentes con el teclado táctil que se desplegará automáticamente. Posteriormente deberá tomar una foto que será vinculada a su cuenta y al contenido que genere durante su navegación en la aplicación. Para ello, deberá presionar el botón ubicado debajo de los campos de texto el cual lo llevará a una la aplicación personalizada de captura de fotos. Una vez tomada y confirmada la foto, la aplicación regresará a la pantalla anterior donde aparecerá dicha foto en lugar del botón presionado anteriormente. Esto se puede observar en la figura 3.3. La foto será utilizada como el Avatar del usuario y se mostrará en las historias y participaciones del mismo. En caso que el usuario desee cambiar la foto sólo deberá repetir el proceso. Una llenados todos los datos el usuario podrá finalizar con el proceso de registro de usuario presionando el botón en la parte inferior de la pantalla. De haber realizado un registro exitoso, la aplicación llevará al usuario a la próxima pantalla. Adicionalmente, un servicio

de correo electrónico enviará automáticamente una notificación a su cuenta de correo dándole la bienvenida a la aplicación.

Figura 3.2. Pantalla de acceso y registro de usuarios.

Vale la pena destacar que aún quedan algunos detalles sobre la información del usuario que completar; localización, aficiones, entre otras. Para ello el usuario deberá accesar a la página web y terminar de realizar su registro. No obstante, la información cumplimentada desde la aplicación móvil es suficiente para que el usuario pueda utilizarla sin ningún tipo de restricciones.

En el caso que el usuario ya se haya registrado anteriormente, sólo deberá ingresar su nombre de usuario y presionar el mismo botón descrito para el caso anterior que lo llevará a la pantalla de *Búsqueda de Historias* que será presentada en el siguiente apartado.

Figura 3.3. Pantalla de acceso y registro de usuarios después de capturar la foto.

3.2 Información de la Historia

Aunque la aplicación es capaz de trabajar con diferentes historias, tal y como se ha mencionado con anterioridad, el objetivo de esta versión alpha es simplemente mostrar el potencial de la plataforma Android e incorporar algunas tecnologías desarrolladas en el MTG. Por esta razón, la aplicación ya tiene seleccionada una historia demo que permite a los usuarios una participación completa, donde se muestran todas las funcionalidades de la aplicación.

Una vez que el usuario se haya registrado o ingresado a la aplicación se iniciará una pantalla donde se presenta la información general de la misma, tal y como se muestra en la figura 3.4. Se puede observar el título de historia en la parte más superior de la pantalla y justo debajo la información de su creador. Se muestra una foto de su avatar, su nombre de usuario y la localidad que éste haya especificado en su cuenta.

Luego se puede observar un apartado de *Settings* o argumento, donde se indica brevemente el argumento de la historia. Este argumento, en texto, fue especificado por su el usuario en la página web al momento de crear la historia. La razón de este apartado es ubicar rápidamente a los demás usuarios en el escenario donde se desarrolla la historia y considerar si desean o no participar en ella. Por último, se muestra una foto y el nombre de los personajes participantes en la historia.

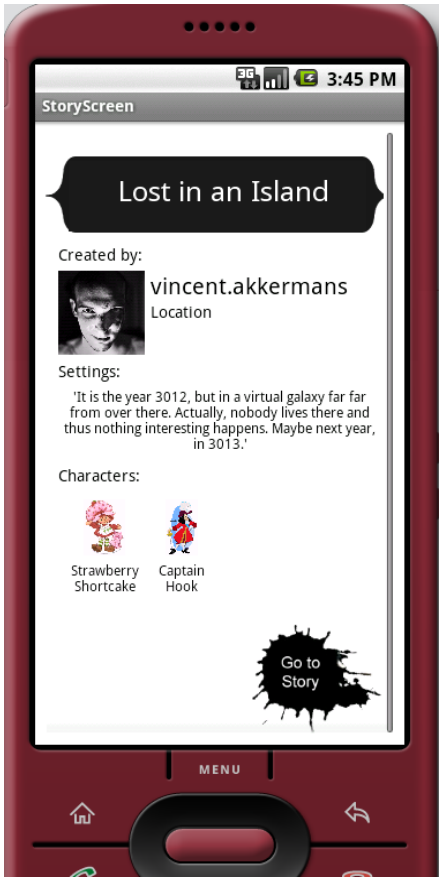

Figura 3.4. Pantalla de información de la historia.

En caso que al usuario le haya gustado el argumento y los personajes, podrá continuar a la siguiente pantalla y escucharla. Para ello, deberá presionar el botón de *Go to Story* en la esquina inferior derecha.

3.3 Contenido de la Historia

Esta es una de las pantallas más importantes de la aplicación. Tal y como se muestra en la figura 3.5, esta pantalla muestra un listado completo de las partes que conforman la historia. Adicionalmente, en cada una de ellas se puede observar el personaje que participa, el usuario que creo dicha parte y dónde se ubicaba geográficamente cuando la creó. Además de la información visual, el usuario podrá escuchar las partes una por una cliqueando directamente sobre ellas o detener su reproducción cliqueando nuevamente.

Figura 3.5. Pantalla de contenido de la historia.

Si así lo desea, el usuario podrá utilizar el botón de Play All, ubicado en la parte inferior de la pantalla, para escuchar toda la historia de manera secuencial. En este caso, para ayudar al usuario a seguir la línea de la historia y ver claramente que personaje está participando, aparecerá un recuadro que muestra al personaje y creador de la parte en

cuestión. La información en este recuadro cambiará dinámicamente según el progreso de la historia. Esto se puede observar en la figura 3.6. Una vez que el usuario no desee seguir escuchando la historia podrá presionar el botón de *Stop* que detendrá la reproducción y cerrará el recuadro.

Figura 3.6. Cuadro de diálogo en la pantalla de contenido de la historia.

En este punto de la aplicación el usuario puede cliquear el botón de *Add* que lo llevará a la próxima pantalla, donde podrá crear nuevas partes y agregarlas a la presente historia. Esto se explicará a continuación en el siguiente apartado.

3.4 Participación en la Historia

Como se acaba de mencionar, es esta la actividad que permitirá al usuario crear nuevas partes y darle continuidad a las historias. Por ello, esta sección puede ser pensada como la parte más creativa y activa de toda la aplicación.

Figura 3.7. Pantalla de participación en la historia

Tal y como se muestra en la figura 3.7, se le presentan al usuario tres botones: *Record, Transform y Upload.* Al presionar el primero de ellos, se abrirá un recuadro donde se muestran los personajes que participan en la historia. Adicionalmente a los personajes aparecerá William Shakespeare, quien es el narrador por defecto en todas las historias. Esto se puede observar en la figura 3.8.

Figura 3.8. Cuadro de personajes de la pantalla de participación en la historia.

Al seleccionar un personaje se cerrará el cuadro y se iniciará un servicio de grabación que registrará la voz del usuario y la asociará a dicho personaje en la historia. Una vez el usuario haya terminado de grabar lo que será una nueva parte en la historia, deberá presionar nuevamente el botón de *Record*. Esto detendrá la grabación y creará un nuevo elemento en la lista de partes. Cada una de las grabaciones, partes, o elementos en la lista, muestra información sobre el usuario, el personaje de la historia, y la ubicación geográfica del usuario al momento de finalizar la grabación. Esto se puede observar en la figura 3.9.

Figura 3.9. Pantalla de participación en la historia con un nuevo elemento.

Para escuchar cada una de las partes que han sido grabadas, el usuario sólo deberá presionar el elemento en la lista. Para detener la reproducción se deberá presionar nuevamente el elemento. En el caso que el usuario haya cometido un error al momento de grabar la voz o simplemente no esté satisfecho con el resultado, podrá borrar esa parte manteniendo presionado dicho elemento durante un par de segundos.

Una vez que el usuario haya finalizado de grabar nuevas partes en la historia podrá presionar el botón de *Transform* para iniciar el servicio de transformación de voz asociado a cada uno de los personajes. Esta asociación fue indicada por el usuario creador de los personajes al momento de crear la historia en la web. Como se muestra en la figura 3.10, esta acción mostrará un cuadro indicando al usuario que la transformación se está llevando a cabo. Una vez finaliza, se le presentará al usuario un mensaje indicando el estatus de dicha transformación. Para escuchar, pausar y borrar cualquier grabación, el usuario deberá proceder de la misma manera que lo hizo anteriormente.

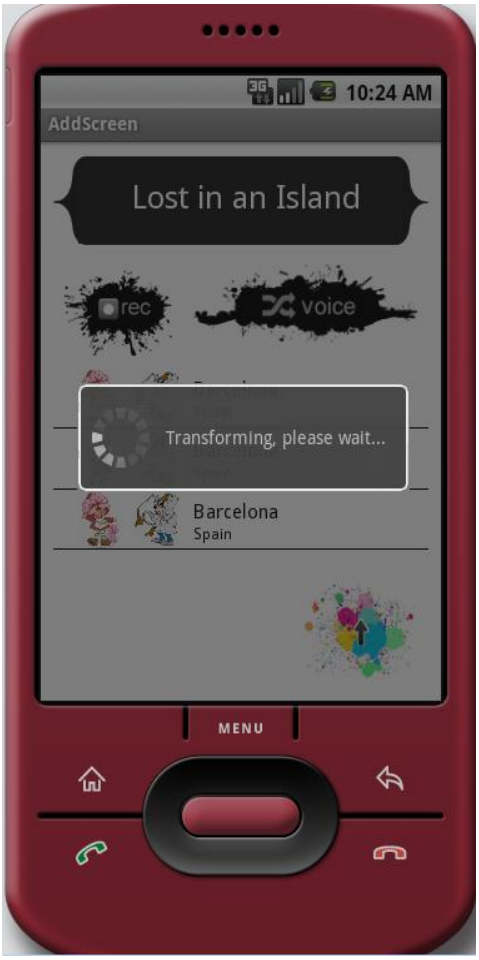

Figura 3.10. Transformación de voz en la pantalla de participación en la historia.

En el caso que haya ocurrido algún error en la transformación, el usuario podrá iniciarla nuevamente. Por otra parte, en el caso de una transformación exitosa el usuario podrá presionar el botón de *Upload* para agregar las nuevas partes a la historia. Una vez terminado este proceso los elementos de la lista desaparecerán y podrán ser escuchadas por cualquier usuario desde la web. En la figura 3.11 se puede observar la pantalla de participación en la historia durante el proceso de subida de las nuevas partes a la web.

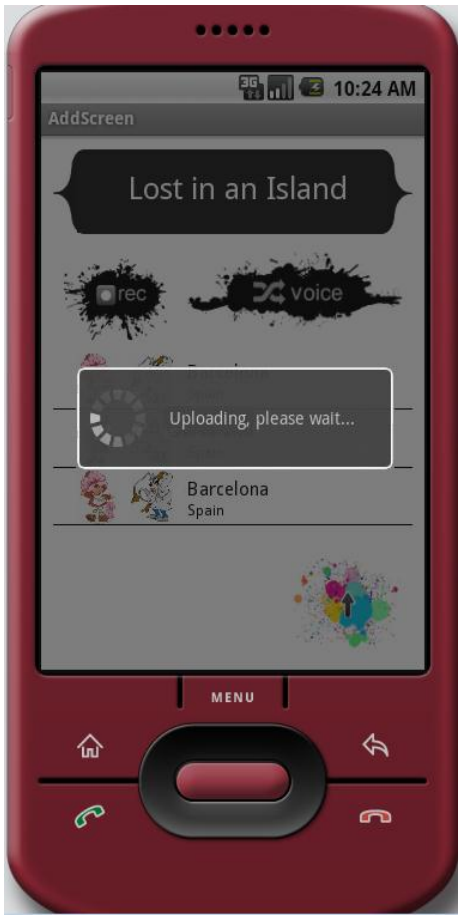

Figura 3.11. Subida de las nuevas partes de una historia.

Capítulo 4

Implementación de las Actividades

Este capítulo explicará, pantalla por pantalla, cómo se implementaron las diferentes funcionalidades de la aplicación. Se mostrarán las funciones encargadas de realizar las diferentes peticiones al servidor del MTG y se explicará en detalle la declaración y configuración de la GUI. Del mismo modo, se presentarán aspectos de implementación que aunque trasparentes al del usuario, son igualmente importantes para el funcionamiento de la aplicación. También se mencionarán algunos aspectos de diseño y tomas de decisiones importantes respecto a las APIs de la plataforma que se utilizaron.

Este capítulo es de suma importancia para comprender el funcionamiento de la aplicación más allá desde el punto de vista usuario, sino desde un punto de vista más técnico y pensado en desarrolladores. De esta manera se comprenderán las implicaciones y competencias de trabajar bajo la plataforma Android.

4.1 Acceso y Registro de Usuarios

Tal y como se mencionó en el capítulo anterior, esta es la primera pantalla que se le mostrará al usuario al ingresar en la aplicación. En ella se podrá registrar y acceder a la misma para posteriormente participar en la historia. En primer lugar, la implementación de

la GUI se realizó declarando los diferentes *Layouts* y *Views* en un documento XML que posteriormente sería controlado a través de las clases Java pertinentes.

```
28    /** Called when the activity is first created. */
29    @Override
30    public void onCreate(Bundle savedInstanceState) {
31        super.onCreate(savedInstanceState);
32        setContentView(R.layout.login);
33
34        Login = (EditText)this.findViewById(R.id.Login_edit);
35        Email = (EditText)this.findViewById(R.id.Email_edit);
36
37        ImageButton Submit = (ImageButton) findViewById
  (R.id.Sumbit);
38        Submit.setOnClickListener(new ImageButton.OnClickListener
  (){
39
40                @Override
41                public void onClick(View arg0) {
42
43                    getEditText();
44                    try {
45                       sendUserInfo();
46                    } catch (UnsupportedEncodingException e) {
47                       e.printStackTrace();
48                    }
49                    Intent intent = new Intent(Login.this,
  StoryScreen.class);
50                    startActivity(intent);
51                }
52        });
53
54        ImageView Participant = (ImageView) this.findViewById
  (R.id.ParticipantPhoto);
55
56        if (TakePhoto.mBitmap != null){
57
58            Participant.setImageBitmap(TakePhoto.mBitmap);
59
60        }
61
62        Participant.setOnClickListener(new
  ImageButton.OnClickListener(){
63
64            @Override
65            public void onClick(View arg0) {
66
67                Intent intent = new Intent(Login.this,
  TakePhoto.class);
68                startActivity(intent);
69
70            }
71        });
72    }
```

Figura 4.1. Método onCreate de la clase Login.

En la figura 4.1 se puede observar la declaración del método *onCreate()*. Este método extiende la clase *Activity*, que como se mencionó en capítulos anteriores es el componente de la plataforma Android para gestión de las pantallas y vistas. En este método se puede ver la llamada *setContentView* que se encarga de dibujar por pantalla los recursos gráficos declarados en el documento XML para esta actividad. En este caso

se pasó como parámetro el documento login.xml. Para mayor detalle este y demás documentos XML se muestran en el Anexo D. Posteriormente se inicializan separadamente los recursos gráficos del documento que llevarán a cabo alguna acción para poder ser gestionados mediante código Java.

Las instancias *Login* e Email son de la clase *EditText*, la cual representa los campos de edición de texto en los cuales el usuario ingresará sus datos para realizar el registro o acceso a la aplicación según sea el caso. Luego se puede observar la instancia *Submit* de la clase *ImageButton*, la cual representa el botón de ingreso mostrado en la parte inferior de la pantalla en la figura 3.1. En este recurso se declaró un *onClickListener* para capturar la acción del usuario al presionar dicho botón. Esto genera una interrupción y ejecuta el código dentro del método *onClick*, el cual tomará los datos de los campos de edición de texto *Login* e Email. Seguidamente ejecutará el método *sendUserInfo*, que se explicará en mayor detalle en este apartado y encargado de realizar la petición al servidor y finalmente llevará al usuario a la próxima pantalla mediante un *Intent*. Estos últimos, tal y como se explicó en capítulos anteriores, se encargan se unir y ejecutar las actividades de la aplicación.

Por último se muestra la declaración *Participant*, la cual es una instancia de la clase *ImageView* que mostrará la foto del usuario que se está registrando e iniciará la pantalla de captura de imagen al cliquear sobre la misma. Para el último caso también se declaró un *onClickListener* que inicia un *Intent* a la actividad *TakePhoto*.

Volviendo al tema del envío de la información del usuario para el registro y acceso a la aplicación, la figura 4.2 muestra el método *sendUserInfo*. En este se comenzó por declarar un cliente HTTP, llamado *client*, el cual llevará a cabo la petición HTTP POST al servidor del MTG ubicado en la URL declarada en el método. Seguidamente se creó una instancia de la clase File con el fichero que contiene la foto capturada por el usuario. Este es un detalle importante ya que obliga el uso de la librería Java de HTTP MIME para poder realizar peticiones HTTP POST con tipo de media *multipart/form-data*, necesario para enviar datos binarios.

Los métodos necesarios para construir el cuerpo de la petición se muestran en las líneas subsiguientes dentro de *sendUserInfo*. Primero se declaran los campos del cuerpo que

contienen el fichero de la foto, el nombre y dirección de correo electrónico del usuario que luego son agregados al cuerpo de la petición, la cual es finalmente ejecutada.

```
82    public void sendUserInfo() throws UnsupportedEncodingException
  {
83
84        AbstractHttpClient client = new DefaultHttpClient();
85        String URL = "http://mtg105.upf.es/api/tales/accounts";
86        File file = new File
  (Environment.getExternalStorageDirectory()+"/photo.jpeg");
87        FileBody photo = new FileBody(file);
88        StringBody username = new StringBody(user_login);
89        StringBody email = new StringBody(user_email);
90        HttpPost POST = new HttpPost(URL);
91        MultipartEntity reqEntity = new MultipartEntity();
92        reqEntity.addPart("username", username);
93        reqEntity.addPart("photo", photo);
94        reqEntity.addPart("email", email);
95        POST.setEntity(reqEntity);
96
97        try {
98
99            client.execute(POST);
100
101       } catch (Exception e) {
102
103           e.printStackTrace();
104       }
105   }
106 }
```

Figura 4.2. Método sendUserInfo de la clase Login.

4.2 Información de la Historia

La actividad que se explicará a continuación es de carácter meramente informativo. En ella el usuario podrá ver la información más importante acerca de la historia; su título, su creador, el entorno donde se desenvuelve y los personajes que en ella participan.

Para poder mostrar toda la información por pantalla primero se tuvieron que obtener los datos pertinentes mediante una petición al servidor. El código de la misma se puede ver en la figura 4.3. El método *getStoryData* pertenece a la clase *DataRetriever* del único paquete de clases Java de la aplicación.

```
43    public static void getStoryData(int story_id) throws
  IllegalStateException, ClientProtocolException, IOException,
  JSONException{
44
45        HttpClient httpclient = new DefaultHttpClient();
46        String URL = server + story_url + story_id;
47        HttpGet GET_story = new HttpGet(URL);
48        InputStream response = httpclient.execute
  (GET_story).getEntity().getContent();
49        String result = StreamToString(response);
50        storydata = new JSONArray(result).getJSONObject(0);
51
52        //Story Title
53        Story.title = storydata.getString("title");
54
55        //Story Language
56        Story.language = storydata.getString("language");
57
58        //Story Settings
59        Story.setting = storydata.getString("setting");
60
61        //Story ID
62        Story.id = storydata.getInt("id");
63
64        //Story Creator's ID
65        Story.creator.id = storydata.getJSONObject
  ("created_by").getInt("id");
66
67        //Story Creator's Name
68        Story.creator.name = storydata.getJSONObject
  ("created_by").getString("username");
69
70        //Story Creator's Photo
71        String creator_avatar_url = media_url +
  storydata.getJSONObject("created_by").getJSONObject
  ("profile").getString("avatar");
72        HttpClient creator_client = new DefaultHttpClient();
73        HttpGet GET_creator = new HttpGet(creator_avatar_url);
74        InputStream creator_response = creator_client.execute
  (GET_creator).getEntity().getContent();
75        Story.creator.avatar = Drawable.createFromStream
  (creator_response, "photo");
76
```

Figura 4.3. Método getStoryData de la clase DataRetriever (I).

Como se puede observar, este método posee *story_id* como parámetro. Este número es el identificador de la historia, único en la base de datos del servidor. Para realizar la petición HTTP se creó un nuevo cliente de la misma forma que en la actividad anterior, sólo que en este caso se realizó una petición GET para obtener el recurso ubicado en la URL *http://mtg105.upf.es/api/tales/stories/[id_de_la_historia]*. Por razones de simplicidad al momento de realizar cambios, esta URL fue separada en diferentes objetos de tipo *String*.

Una vez que el servidor haya recibido y procesado satisfactoriamente la petición del cliente, devuelve una respuesta en texto en formato JSON. En diseños anteriores se pensó enviar las respuestas en XML, pero por la simplicidad de lectura e implementación de analizadores de texto JSON en Java, se optó por este último como formato. En la siguiente figura se puede observar un ejemplo de respuesta del servidor en formato JSON.


```
[
    {
        "language": "English",
        "title": "Once upon a time...",
        "created_by": {
            "username": "vincent.akkermans",
            "profile": {
                "avatar": "tales/story_tellers/me_face_1.jpg"
            },
            "id": 5
        },
        "participants": [
            {
                "username": "vincent.akkermans",
                "id": 5,
                "avatar": "tales/story_tellers/me_face_1.jpg"
            }
        ],
        "setting": "It is the year 3012, but in a virtual galaxy far far from
                    over there. Actually, nobody lives there and thus nothing
                    interesting happens. Maybe next year, in 3013.",
        "characters": [
            {
                "id": 3,
                "avatar": "tales/story_characters/PinocchioCover.jpg",
                "full_name": "Pinochio",
                "voice_type": "ibc_male02"
            },
            {
                "id": 4,
                "avatar": "tales/story_characters/cookie1.jpg",
                "full_name": "Cookiemonster",
                "voice_type": "MaleToFemale2"
            }
        ],
        "id": 7
    }
]
```


Figura 4.4. Respuesta del servidor a una petición de información de la historia.

Esta respuesta posee toda la información necesaria para que el usuario tenga una idea general de la historia. En el formato JSON cada objeto esta comprendido entre llaves y cada uno de ellos puede tener varios campos separados por comas. También es posible tener arreglos de objetos comprendidos dentro de corchetes y separados por comas. En este caso el servidor responde un arreglo de historias con un solo elemento.

Vale la pena destacar que al igual que las historias, los usuarios y personajes también poseen un identificador único en la base de datos del servidor. Otro aspecto importante es el avatar de cada uno de los anteriores el cual esta representado por la URL del imagen.

Una vez obtenida la respuesta del servidor es necesario recuperar cada uno de los campos del archivo JSON y pasarlos a los objetos de las clases pertinentes para poder ser interpretadas por el código Java. En el caso de los campos de texto como el nombre de la historia, usuarios y personajes, esta acción se lleva a cabo de una manera considerablemente simple. Por otra parte, para obtener las imágenes de los avatares fue necesario realizar peticiones HTTP GET para cada una de ellas indicando la URL obtenida en la petición anterior del repositorio de recursos media *http://mtg105.upf.es/site_media/*. En la figura 4.5 se puede ver el resto del método *getStoryData* y cómo son obtenidos el resto de recursos.

```
77          //Stroy Characters
78          numcharacters = storydata.getJSONArray
    ("characters").length();//+1 for the Narrator
79
80          //Add characters to ArrayList
81          int position = 0;
82
83          for (position = 0; position <
    DataRetriever.numcharacters ; position++){
84
85              Character character = new Character();
86              Story.characters.add(character);
87
88              //ID
89              Story.characters.get(position).id =
    storydata.getJSONArray("characters")
90                  .getJSONObject(position).getInt("id");
91
92              //Name
93              Story.characters.get(position).name =
    storydata.getJSONArray("characters")
94                  .getJSONObject(position).getString("full_name");
95
96              //Voice
97              Story.characters.get(position).voice =
    storydata.getJSONArray("characters")
98                  .getJSONObject(position).getString("voice_type");
99
100             //Avatar
101             String character_avatar_url = media_url +
    storydata.getJSONArray("characters")
102                 .getJSONObject(position).getString("avatar");
103             HttpClient character_client = new DefaultHttpClient();
104             HttpGet GET_character = new HttpGet
    (character_avatar_url);
105             InputStream character_response =
    character_client.execute(GET_character).getEntity().getContent();
106             Story.characters.get(position).avatar =
    Drawable.createFromStream(character_response, "photo");
107
108         }
```

Figura 4.5. Método getStoryData de la clase DataRetriever (II).

Una vez explicada la función del método *getStoryData*, se presentará la clase *StoryScreen* que extiende a la clase *Activity* y se encargará de mostrar pon pantalla la información de la historia. En la figura 4.6 se muestra el método *onCreate* de dicha clase.

Primeramente se puede observar el objeto entero *story* que representa el identificador de la historia a mostrar. Aunque la aplicación, tal y como se mencionó en el capítulo anterior, puede dar soporte a múltiples historias, para su efecto práctico concreto se fijó una historia específica. En esta caso, la historia con identificador 124.

```
18      final static int story = 124;
19      /** Called when the activity is first created. */
20      @Override
21      public void onCreate(Bundle savedInstanceState) {
22          super.onCreate(savedInstanceState);
23          setContentView(R.layout.story);
24
25          try {
26
27              DataRetriever.getStoryData(story);
28
29              TextView StoryTitle = (TextView) this.findViewById
    (R.id.StoryTitle);
30              StoryTitle.setText(DataRetriever.getTitle());
31
32              TextView CreatorName = (TextView) this.findViewById
    (R.id.CreatorName);
33              CreatorName.setText(DataRetriever.getCreatorName());
34
35              TextView Settings = (TextView) this.findViewById
    (R.id.SettingsInfo);
36              Settings.setText("'"+DataRetriever.getSetting()+"'");
37
38              ImageView CreatorAvatar = (ImageView)
    this.findViewById(R.id.CreatorPhoto);
39              CreatorAvatar.setImageDrawable
    (DataRetriever.getCreatorAvatar());
40
41              GridView Characters = (GridView) findViewById
    (R.id.CharacterLayout);
42              Characters.setAdapter(new CharacterAdapter(this));
43
44              ImageButton GoToStory = (ImageButton) findViewById
    (R.id.GoToStory);
45              GoToStory.setOnClickListener(new View.OnClickListener
    () {
46
47                  @Override
48                  public void onClick(View v) {
49
50                      Intent intent = new Intent(StoryScreen.this,
    PartScreen.class);
51                      startActivity(intent);
52
53                  }
54              });
55
56          } catch (Exception e) {
57              e.printStackTrace();
58          }
59      }
```

Figura 4.6. Método onCreate de la clase StoryScreen

Al igual que en todas las actividades de la plataforma Android el método más importante es el método *onCreate*, el cual será ejecutado apenas se cree la historia. En primer lugar se puede observar la llamada al método *getStoryData* antes explicado. Este ejecutará las peticiones necesarias al servidor y obtendrá los recursos necesarios para mostrar por

pantalla. Posteriormente se encuentran las declaraciones de las diferentes vistas del *layout* a las cuales se les pasan los recursos obtenidos anteriormente.

Vale la pena destacar el caso del objeto *Characters* de tipo *GridView* el cual se encargará de dibujar los avatares y nombres de cada uno de los personajes participantes en la historia ordenados a manera de cuadrícula, o *grid*. Se puede notar que en *Characters* no se ejecutan simplemente los métodos que establecen los recursos gráficos como en las otras vistas, sino que se le pasa un adaptador personalizado llamado *CharacterAdapter*.

```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
  android"
2     android:id="@+id/CharacterTemplate"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:orientation="vertical"
6     android:gravity="center_horizontal"
7     android:background="#ffffff">
8
9     <ImageView
10        android:id="@+id/CharacterAvatar"
11        android:src="@drawable/icon"
12        android:padding="5dip"
13        android:adjustViewBounds="true"
14        android:scaleType="fitCenter"
15        android:layout_height="60dip"
16        android:layout_width="60dip"
17        android:clickable="true"/>
18
19    <TextView
20        android:id="@+id/CharacterName"
21        android:text="Character Name"
22        android:layout_width="wrap_content"
23        android:layout_height="wrap_content"
24        android:gravity="center"
25        android:paddingBottom="5dip"
26        android:textSize = "12sp"
27        android:textColor = "#000000"/>
28
29 </LinearLayout>
```

Figura 4.7. Archivo XML de plantilla de personajes.

El uso de un adaptador es necesario ya que se desean mostrar tanto el avatar como el nombre de los personajes, es decir, un elemento de vista con dos tipos de recursos. Para ello se declaró en la carpeta de layout un archivo XML llamado *carácter.xml* que serviría de plantilla para este tipo de objetos dentro del *GridView*. Este archivo se puede observar en la figura 4.7.

Una vez declarada la plantilla es necesario indicarle al adaptador con qué recursos debe *inflarla*. La declaración de la clase del adaptador se muestra en la figura 4.8.

```
61    public class CharacterAdapter extends BaseAdapter {
62
63        private Context mContext;
64
65        public CharacterAdapter(Context c) {
66            mContext = c;
67        }
68
69        public View getView(int position, View convertView,
  ViewGroup parent) {
70
71            View view;
72
73            if (convertView == null) {
74
75                LayoutInflater inflater = (LayoutInflater)
  mContext.getSystemService(
76                    Context.LAYOUT_INFLATER_SERVICE);
77                view = inflater.inflate(R.layout.character, null);
78
79                ImageView imageView = (ImageView)
  view.findViewById(R.id.CharacterAvatar);
80                imageView.setImageDrawable
  (DataRetriever.getCharacterAvatars(position));
81
82                TextView textView = (TextView) view.findViewById
  (R.id.CharacterName);
83                textView.setText(DataRetriever.getCharacterNames
  (position));
84
85            } else {
86
87                view = convertView;
88            }
89
90            return view;
91
92        }
93
94        public int getCount() {
95
96            return DataRetriever.numcharacters;
97        }
```

Figura 4.8. Adaptador para la vista de personajes de la clase StoryScreen.

La clase *CharacterAdapter* posee dos métodos fundamentales, *getView* y *getCount*. El primero de ellos se encarga de retornar la vista al objeto *GridView* que lo llama. En este método se inicializa el servicio del sistema que infla la vista, al cual posteriormente se le indica la plantilla sobre la cual se deberán dibujar los recursos. El método *getView*, además de parámetros de vista, posee un parámetro de posición el cual el indica cuál objeto dentro de la cuadrícula se está tratando. Aunque no es este caso, este parámetro

es de suma importancia cuando se desea discriminar un objeto en específico dentro de los objetos del adaptador y llevar a cabo alguna acción determinada según el objeto. Más adelante en este capítulo se mostrará este tipo de casos. No obstante, *CharacterAdapter* necesita este parámetro para dibujar los recursos en el lugar correspondiente dentro del *GridView*.

Después de haber establecido la plantilla, es necesario inicializar cada uno de los elementos en la misma y asignarles un valor. Como se observó en el método *getStoryData*, los recursos referentes a los personajes fueron guardados en un arreglo de la clase *Character*. De esta manera sólo es necesario indicar el tipo de recurso deseado dentro de esta clase y su posición dentro del arreglo.

El segundo método del adaptador, *getCount*, indica el número de elementos que se desean inflar. Esto hará que el método *getView* realice todas iteraciones necesarias para dibujar dichos elementos. En este caso se indicó que se debía iterar tantas veces como personajes haya.

4.3. Contenido de la Historia

En esta pantalla se le presentará al usuario un listado de las partes que comprenden la historia, los segmentos de audio que representan la participación de cada uno de los personajes en la historia. Tal como se explicó en el capítulo anterior, el usuario podrá reproducir y detener cada una de ellas. Adicionalmente, se le de dará la oportunidad de reproducirlas todas de manera secuencial con la ayuda de una ventana de diálogo que cambiará dinámicamente dependiendo de la parte que esté reproduciendo en un momento dado.

Al igual que en la pantalla anterior, es necesario hacer una petición al servidor para obtener la información de cada una de las partes de la historia. En la clase *DataRetriever* se implementó un método para esta función, *getPartData*. Tal y como se muestra en la figura 4.9, se debe realizar una petición HTTP GET a la URL donde se encuentran los recursos de las partes de la historia deseada, cuyo identificador se pasa como parámetro en el método. EL servidor responde a esta petición con un documento en formato JSON con un arreglo de objetos donde cada uno de ellos contiene la información de cada una

de las partes de la historia. Un ejemplo de este tipo de respuesta se muestra en la figura 4.9.

```
158     public static void getPartData(int story_id) throws
    IllegalStateException, ClientProtocolException, IOException,
    JSONException{
159
160         HttpClient httpclient = new DefaultHttpClient();
161         String URL = server + story_url + story_id + "/parts";
162         HttpGet GET = new HttpGet(URL);
163         InputStream response = httpclient.execute(GET).getEntity
    ().getContent();
164         String result = StreamToString(response);
165         partsdata = new JSONArray(result);
166
167         numparts = partsdata.length();
168
169         int part_number = 0;
170         for (part_number = 0 ; part_number < numparts ;
    part_number++){
171
172             Part part = new Part();
173             parts.add(part);
174
175             //Characters' ID
176             parts.get(part_number).character =
    partsdata.getJSONObject(part_number).getInt("character");
177
178             //Participants' ID
179             parts.get(part_number).creator =
    partsdata.getJSONObject(part_number).getInt("creator");
180
181             //Parts' Keys
182             parts.get(part_number).key = partsdata.getJSONObject
    (part_number).getString("key");
183
184             //City
185             parts.get(part_number).city = partsdata.getJSONObject
    (part_number).getString("city");
186
187             //Country
188             parts.get(part_number).country =
    partsdata.getJSONObject(part_number).getString("country");
189         }
```

Figura 4.9. Método getPartData de la clase DataRetriever (I).

La respuesta del servidor es posteriormente recopilada objeto por objeto, obteniendo cada uno de los campos y asignando sus valores a los objetos Java correspondientes tal como se muestra en la figura 4.10. Se puede observar que esta respuesta del servidor no contiene toda la información necesaria de cada una de las partes. Datos como los avatares y nombres del creador y personajes se omitieron en la respuesta ya que estos se

obtuvieron en la petición de información de la historia. Para recopilar los datos es necesario realizar un mapeo de los identificadores del personaje y creador obtenidos en la petición de las partes con los obtenidos en la petición de información de la historia. Esto se lleva a cabo en la segunda parte de método *getPartData*, mostrada en la figura 4.11.


```
[
  {
    'character': 8,
    'key': u'3b9e6e9e-3dc6-4520-b47c-39efaefcf2bb',
    'creator': 5,
    'city': Barcelona,
    'country': Spain
  },
  {
    'character': 8,
    'key': u'194d9cc5-accf-45a5-84dc-c7c835112e40',
    'creator': 5,
    'city': Barcelona,
    'country': Spain
  },
  {
    'character': 8,
    'key': u'ff41be41-62bc-4749-bdfd-109c33a74cf8',
    'creator': 5,
    'city': Barcelona,
    'country': Spain
  },
  {
    'character': 8,
    'key': u'50125847-d35f-4bff-985a-b47caa8192a9',
    'creator': 5,
    'city': Barcelona,
    'country': Spain
  },
  {
    'character': 4,
    'key': u'005859e1-6ae5-433d-a060-41329269660c',
    'creator': 5,
    'city': Barcelona,
    'country': Spain
  },
  {
    'character': 3,
    'key': u'a0002164-cff8-4403-8ebc-31ebe01c2396',
    'creator': 5,
    'city': Barcelona,
    'country': Spain
  },
  {
    'character': 8,
    'key': u'e1f05d7d-c936-4b3b-bc19-0d3b132b3abe',
    'creator': 5,
    'city': Barcelona,
    'country': Spain
  }
]
```


Figura 4.10. Respuesta del servidor a una petición de las partes de la historia.

Primero se procede a realizar el mapeo del nombre y avatar del personaje. Para ello, se comparan cada uno de los identificadores de personaje obtenidos en la petición de las

partes con los obtenidos en la petición de información de la historia. Al encontrar la concordancia entre ellos se procede a asignar el nombre y avatar al objeto Java correspondiente a esa parte de la historia. Luego se realiza el proceso análogo entre los creadores de la parte y los participantes en la historia.

```
190         //Mapping...
191
192         int story_character = 0;
193         int part_character = 0;
194
195         for (part_character = 0 ; part_character <
    DataRetriever.numparts; part_character++){
196             for (story_character = 0 ; story_character <
    DataRetriever.numcharacters + 1; story_character++){
197                 if (parts.get(part_character).character ==
    Story.characters.get(story_character).id){
198                     parts.get(part_character).character_photo =
    Story.characters.get(story_character).avatar;
199                     parts.get(part_character).charname =
    Story.characters.get(story_character).name;
200                 }
201             }
202         }
203
204         int story_participant = 0;
205         int part_participant = 0;
206
207         for (part_participant = 0 ; part_participant <
    DataRetriever.numparts ; part_participant++){
208             for (story_participant = 0 ; story_participant <
    DataRetriever.numparticipants ; story_participant++){
209                 if (parts.get(part_participant).creator ==
    Story.participants.get(story_participant).id){
210                     parts.get(part_participant).creator_photo =
    Story.participants.get(story_participant).avatar;
211                     parts.get(part_participant).creator_name =
    Story.participants.get(story_participant).name;
212                 }
213             }
214         }
215     }
```

Figura 4.11. Método getPartData de la clase DataRetriever (II).

Como se mostró en el capítulo anterior en la figura 3.4, esta pantalla muestra el título de la historia, las partes que la forman y botones que le permitirán al usuario iniciar y detener la reproducción de las mismas y continuar a la próxima pantalla de la aplicación.

En la figura 4.12 se muestra el método *onCreate* el cual es ejecutado al iniciar la pantalla de contenidos de la historia y sirve para comprender el flujo de ejecución de la aplicación y las diferentes funciones que se llevan a cabo. Al igual que en el resto de las actividades

de la aplicación, primero se debe asignar el archivo XML del layout el cual indica las diferentes vistas que conformarán la pantalla y sus respectivos parámetros.

```
42    public void onCreate(Bundle icicle){
43
44      super.onCreate(icicle);
45      setContentView(R.layout.storyparts);
46
47      try{
48
49          DataRetriever.getPartData(StoryScreen.story);
50
51          TextView StoryTitle = (TextView) this.findViewById
  (R.id.StoryTitle);
52          StoryTitle.setText(DataRetriever.getTitle());
53
54          ListView PartsList = (ListView) findViewById
  (android.R.id.list);
55          PartsList.setAdapter(new PartListAdapter(this));
56
57          ImageButton PlayAllButton = (ImageButton)
  this.findViewById(R.id.PlayButton);
58          PlayAllButton.setOnClickListener(new
  ImageButton.OnClickListener(){
59
60              @Override
61              public void onClick(View arg0){
62                  new PlayAllParts().execute();
63              }
64          });
65
66          ImageButton AddButton = (ImageButton) findViewById
  (R.id.AddButton);
67          AddButton.setOnClickListener(new View.OnClickListener()
  {
68
69                @Override
70                public void onClick(View v) {
71                    try{
72                        HttpClient httpclient = new
  DefaultHttpClient();
73                        String URL = DataRetriever.server +
  DataRetriever.story_url + StoryScreen.story + "/lock";
74                        HttpPost postlock = new HttpPost(URL);
75                        InputStream response = httpclient.execute
  (postlock).getEntity().getContent();
76                        lock = DataRetriever.StreamToString
  (response);
77
78                    }catch(Exception e){
79                        e.printStackTrace();
80                    }
81
82                    Intent intent = new Intent(PartScreen.this,
  AddScreen.class);
83                    startActivity(intent);
84                }
85          });
86      }catch (Exception e){
87          e.printStackTrace();
88      }
89    }
```

Figura 4.12. Método onCreate de la pantalla de contenido de la historia.

Una vista a destacar en esta pantalla es la lista de las partes que llenará la mayor parte de la pantalla y será el foco de atención del usuario al momento de interactuar con la

aplicación. En el método *onCreate* esta lista se encuentra representada por el *PartsList* del tipo *ListView*. Esta clase de la API de la plataforma *Android* representa uno de los tipos de recursos gráficos de mayor importancia en las aplicaciones ya que permite mostrar información y capturar la interacción con el usuario para implementar numerosas funcionalidades. Además, las listas pueden ser personalizadas y contener a su vez otros recursos gráficos en cada uno de sus elementos. En el caso de esta aplicación cada uno de los elementos de la lista posee la imagen del avatar del personaje participante, la foto del usuario creador y al igual que el país y cuidad donde se encontraba el mismo al momento de crear esa parte de la historia. En las figuras 4.13 se muestra el archivo XML que indica el layout de cada uno de los elementos de la lista.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:id="@+id/PartTemplate"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal"
    android:paddingLeft="15dip"
    android:gravity="center"
    android:background="#ffffff">

  <ImageView
       android:id="@+id/CharacterAvatar"
       android:src="@drawable/shakespeare"
       android:paddingLeft="5dip"
       android:paddingRight="5dip"
       android:adjustViewBounds="true"
       android:scaleType="fitCenter"
       android:layout_height="45dip"
       android:layout_width="45dip"
       android:focusable="false"/>

  <ImageView
       android:id="@+id/ParticipantAvatar"
       android:src="@drawable/icon"
       android:paddingLeft="5dip"
       android:paddingRight="5dip"
       android:adjustViewBounds="true"
       android:scaleType="fitCenter"
       android:layout_height="45dip"
       android:layout_width="45dip"
       android:focusable="false"/>

  <LinearLayout
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
       android:orientation="vertical"
       android:gravity="center"
       android:background="#ffffff"
       android:focusable="false">
      <TextView
           android:id="@+id/City"
           android:layout_width="fill_parent"
           android:layout_height="wrap_content"
           android:paddingTop="5dip"
           android:paddingLeft="5dip"
           android:gravity="left"
           android:textSize = "15sp"
           android:textColor = "#000000"
           android:focusable="false"/>
       <TextView
           android:id="@+id/Country"
           android:layout_width="fill_parent"
           android:layout_height="wrap_content"
           android:paddingBottom="5dip"
           android:paddingLeft="5dip"
           android:gravity="left"
           android:textSize = "12sp"
           android:textColor = "#000000"
           android:focusable="false"/>
    </LinearLayout>
</LinearLayout>
```

Figura 4.13. Layout de cada parte en la lista de contenidos de la historia.

Para poder asignar a cada elemento de *PartsList* un *layout* con las características de la figura 4.13 fue necesario implementar un adaptador y asignarlo a este objeto en el método *onCreate*. En este caso se procedió de la misma manera que en el adaptador utilizado en la pantalla anterior para mostrar los personajes.

Los elementos de la lista también pueden responder a acciones de los usuarios. En esta caso esta acción iniciará y detendrá la reproducción de cualquiera de las partes de la lista de contenidos de la historia. Para ello es necesario implementar el método *onListItemClick*

de la clase *ListActivity* el cual generará una interrupción en el hilo de ejecución cuando el usuario realice un clic en alguno de los elementos de la lista. En la figura 4.14 se muestra la implementación de este método para la clase *PartScreen*.

```
91    public void onListItemClick (ListView parent, View v, int
  position, long id){
92
93        if (current_part != position  && mp.isPlaying() == false){
94            try{
95                mp.release();
96                mp = new MediaPlayer();
97                mp.setDataSource(DataRetriever.server +
  DataRetriever.files_url + DataRetriever.getPartKey(position) +"/
  conversion/mp3");
98                mp.prepare();
99                mp.getDuration();
100               mp.start();
101               current_part = position;
102           }catch( Exception e){
103               e.printStackTrace();
104           }
105
106       }else if ((isPlaybackCompleted() || isPaused()) &&
  current_part==position){
107           mp.start();
108
109       }else if (mp.isPlaying() && current_part == position){
110           mp.pause();
111
112       }if (current_part != position  && mp.isPlaying() == true){
113           try{
114               mp.pause();
115               mp.release();
116               mp = new MediaPlayer();
117               mp.setDataSource(DataRetriever.server +
  DataRetriever.files_url + DataRetriever.getPartKey(position) +"/
  conversion/mp3");
118               mp.prepare();
119               mp.getDuration();
120               mp.start();
121               current_part = position;
122           }catch( Exception e){
123               e.printStackTrace();
124           }
125       }
126   }
```

Figura 4.14. onListItemClick de la actividad PartScreen.

Para implementar las acciones en este método se tomaron en cuenta los cuatro casos posibles de interacción. El primero de ellos corresponde a un clic del usuario en un elemento de la lista que no estaba siendo reproducido mientras el reproductor estaba inactivo. En este caso se inicia el reproductor pasando como parámetro la URL de la grabación correspondiente a ese elemento. El segundo caso corresponde al clic de un elemento que fue pausado en la acción anterior o se reproducción finalizó. En este caso se desea reiniciar la reproducción sin tener que inicializar nuevamente al reproductor de

media. El tercer caso corresponde a la acción del usuario sobre un elemento de la lista que se esta siendo reproducido. En ese caso se detiene el reproductor de media. El último caso a considerar ocurre cuando el usuario clica un elemento de la lista mientras otra de las partes esta siendo reproducida en ese momento. Si esto pasa, se detiene la reproducción actual y se inicializa la nueva reproducción.

Como se mencionó anteriormente además de mostrar información por pantalla esta actividad permite reproducir de manera secuencial, y con ayuda de una ventana gráfica, todas las partes de la historia. Para realizar esta acción el usuario deberá presionar el botón de *Play All* declarado en el método *onCreate* bajo el nombre *PlayAllButton*. En la instancia de este botón se declara un *onClickListener* para capturar la interacción de usuario y generar una interrupción que ejecutará el código de su método *onClick*. En este se encuentra la declaración del método *execute* de la clase *PlayAllParts* que extiende la clase *AsyncTask* explicada a continuación y presentada en la figura 4.15.

```
180    private class PlayAllParts extends AsyncTask <Drawable,
   Drawable, Object> {
181
182        protected void onPreExecute(){
183            play = true;
184            storytellingdialog();
185        }
186
187        protected Object doInBackground(Drawable...drawables ) {
188            try{
189                int i;
190
191                for (i = 0; i < DataRetriever.numparts; i++){
192                    if ( play == true){
193
194                        playchar =
   DataRetriever.getPartCharacterDrawable(i);
195                        playparticipant =
   DataRetriever.getPartParticipantDrawable(i);
196                        playcharname =
   DataRetriever.getPartCharacterName(i);
197                        playparticipantname =
   DataRetriever.getPartParticipantName(i);
198
199                        publishProgress(playchar,
   playparticipant);
200
201                        mp.release();
202                        mp = new MediaPlayer();
203                        mp.setDataSource(DataRetriever.server +
   DataRetriever.files_url + DataRetriever.getPartKey(i) +"/
   conversion/mp3");
204                        mp.prepare();
205                        mp.getDuration();
206                        mp.start();
207
208                        while (mp.isPlaying()){
209                            Thread.sleep(300);
210                        }
211                    }
212                }
213
214            }catch (Exception e){
215                e.printStackTrace();
216            }
217            return null;
218        }
219
220        protected void onProgressUpdate(Drawable... progress) {
221            dialogInflator();
222        }
223    }
```

Figura 4.15. Implementación de AsyncTask para la reproducción las partes de la historia.

La plataforma *Android* ofrece una clase llamada *AsyncTask* para que permite al desarrollador llevar a cabo programación multi-hilo de una manera transparente. Esta clase posee métodos para llevar a cabo acciones en un hilo en el fondo y evitar interrumpir la ejecución del hilo de la interfaz gráfica. Esto permite realizar tareas de larga computación mientras el usuario interactúa de manera normal con los botones y el resto de recursos de la GUI. Adicionalmente, la clase *AsyncTask* posee métodos que se

ejecutan en el hilo de la interfaz gráfica permitiendo mostrar dinámicamente datos por pantalla a medida que estos se obtienen en el hilo del fondo.

El primer método que se llama al hace la llamada *execute* de la clase *PlayAllParts* es el método *onPreExecute*. Las declaraciones contenidas en él se llevan a cabo en el hilo de la interfaz gráfica. En este caso particular *onPreExecute* ejecuta el método *storytellingdialog* encargado de iniciar la ventana gráfica que contendrá el avatar e imagen del personaje y creador de la parte de la historia que se esté reproduciendo en ese momento. Al finalizar la ejecución de *onPreExecute*, *AsyncTask* llamará automáticamente al método *doInBackground* donde se realizan todas las tareas que deben ejecutarse en el hilo del fondo. Este método se encarga de ir a través de la lista de partes y obtener los datos necesarios del personaje y creador y asignarlos a las variables de la ventana de *storytellingdialog*. La llamada al método *publishProgress* genera una interrupción y ejecuta en el hilo de la interfaz gráfica al método *onProgressUpdate* encargado de dibujar por pantalla los nuevos datos obtenidos. Adicionalmente, el método *doInBackground* crea una instancia de un reproductor de la clase *MediaPlayer* de la plataforma, a la cual le pasa como parámetro la URL con la ubicación de grabación de dicha parte. Una vez finalizada la inicialización y preparación se procede a reproducir el archivo de audio. Esta serie de acciones se realizan secuencialmente con cada una de las partes de la historia.

Por último, esta el botón que lleva al usuario a la pantalla de participación en la historia declarado en el método *onCreate* como *AddButton*. Al tratarse de una aplicación multiusuario, donde varias personas podrían estar interactuando con la misma historia, es necesario establecer reglas de acceso en la participación activa de la misma. Aunque no hay problemas si más de una persona escucha las partes de la historia al mismo tiempo, si se le permitiera a cualquier usuario agregar nuevas partes a la historia en cualquier momento, podrían ocurrir conflictos de concurrencia y sería posible la existencia de diferentes versiones de una misma historia. Por ello, cuando el usuario solicita a la aplicación que le lleve a la pantalla de participación en a historia se genera una petición HTTP POST donde el usuario solicita un bloqueo temporal de la historia para su uso. De esta manera se evita que más de un usuario modifique la historia al mismo tiempo.

4.4 Participación en la historia

En las pantallas anteriores se presentó lo que puede considerarse como la parte pasiva de la aplicación y no es, sino hasta este punto, donde el usuario activamente participará en la historia y le dará continuidad a los relatos. Esta pantalla puede ser vista como la más importante y la que reúne la mayor cantidad de funcionalidades de toda la aplicación. Este apartado intentará presentar cómo se implementaron cada una de ellas y los factores que se tomaron en cuenta en la toma de decisiones correspondientes.

A lo largo de este capítulo se mostraron los diferentes elementos de la GUI y los archivos XML que la representa, se mostraron los adaptadores necesarios para dibujar dinámicamente de las diferentes vistas al igual que la implementación necesaria para la reproducción de audio de las partes de historia. Por esta razón no se presentarán nuevamente funciones de este tipo a pesar de estar presentes en esta pantalla. Por otra parte, este apartado mostrará cómo se implementaron funcionalidades que, aunque transparentes para el usuario, son de gran importancia para el desempeño de la aplicación. Entre estas se pueden mencionar la función de geolocalización, el grabador de audio en formato WAVE, la rutina de interacción con el servicio de transformación de voz y por último la subida de las nuevas partes al servidor de la aplicación.

La actividad *AddScreen* es la encargada de mostrar por pantalla los elementos gráficos de esta vista y desde esta donde se realizan las llamadas a las funciones mencionadas anteriormente.

Ya que esta aplicación podría ser usada en cualquier parte del mundo con conexión a Internet, fue deseable destacar el aspecto global y ubicuo de este tipo de tecnologías. Por ello, se implementó una función que captura la localización del usuario al momento de crear una parte en la historia.

Para iniciar el servicio de geolocalización se hace una llamada al método *startGPS* de la clase *MyGPS*. Esta llamada se realiza dentro del método *onCreate* de la actividad *AddScreen* de manera que el gestor de localización de la plataforma comience a capturar información de la ubicación del dispositivo en el momento que el usuario pueda comenzar a grabar las nuevas partes de la historia. El método *startGPS* se muestra en la figura 4.16.

```
27    public void startGPS(Context context){
28        mylocationmanager = (LocationManager)
  context.getSystemService(Context.LOCATION_SERVICE);
29        mylocationlistener = new MyLocationListener();
30        mylocationmanager.requestLocationUpdates(mylocationprovider, 0, 1000,
31                mylocationlistener); //updated if the user moves 1km.
32
33    }
```

Figura 4.16. Método *startGPS()*.

En primer lugar se debe crear una instancia del LocationManager o gestor de localización del sistema. Debido a que la posición del usuario puede cambiar, es necesario actualizar la información de localización del dispositivo. Sin embargo, es ineficiente realizar esta actualización de manera síncrona ya que se podrían realizar peticiones de localización mientras el usuario se mantenga estático. El gestor de localización del sistema ofrece un método de actualización que permite generar interrupciones asíncronas según parámetros de localización establecidos. En este caso se decidió que estas interrupciones se generaran cada vez que el usuario se mueva 1km ya que sólo será necesario obtener la ciudad y país donde se encuentra ubicado el usuario. Cada vez que se generan una de estas interrupciones se llama al método *onLocationChanged* de la clase *LocationListener* mostrado en la figura 4.17.

```
35    private class MyLocationListener implements LocationListener {
36
37        public void onLocationChanged(Location loc) {
38            if (loc != null) {
39                coordenates = new GeoPoint((int) (loc.getLatitude() * 1E6),
40                        (int) (loc.getLongitude() * 1E6));
41                Log.v("Location", coordenates+"");
42            }else{
43                loc = mylocationmanager.getLastKnownLocation(mylocationprovider);
44                coordenates = new GeoPoint((int) (loc.getLatitude() * 1E6),
45                        (int) (loc.getLongitude() * 1E6));
46                Log.v("Location", coordenates+"");
47            }
48            Geocoder geoCoder = new Geocoder(
49                    mContext, Locale.getDefault());
50
51            try {
52
53                addresses = geoCoder.getFromLocation(
54                        coordenates.getLatitudeE6()  / 1E6,
55                        coordenates.getLongitudeE6() / 1E6, 1);
56                Address address = addresses.get(0);
57                city = address.getLocality();
58                country = address.getCountryCode();
59
60            } catch (Exception e) {
61                e.printStackTrace();
62            }
63        }
```

Figura 4.17. Método *onLocationChanged()* de la clase *MyGPS*.

Este método hace uso de la clase *GeoPoint* de la API de mapas de la plataforma para obtener la longitud y latitud de la localización del usuario mediante el *LocationProvider*, o proveedor de localización. En este caso la instancia del proveedor de localización se inicializó para trabajar por triangulación de celdas telefónicas. Este método es menos preciso que la localización por GPS, pero suficiente para los efectos prácticos de la aplicación. Una vez obtenidas la longitud y latitud se extrae la información de la ciudad y país correspondientes.

Otra de las funcionalidades de esta pantalla es el grabador de audio. La plataforma *Android* posee una API llamada *MediaRecorder* para la grabación de audio y vídeo de una manera simple y efectiva. Sin embargo, uno de los requerimientos de la aplicación es que las partes sean grabadas en formato WAVE, el cual es un formato de codificación no aceptado por la plataforma *Android*. Por ello, se tuvo que implementar un grabador PCM mediante la clase *AudioRecorder* y posteriormente agregar la cabecera WAVE al fichero en formato RAW, o crudo. Los parámetros seleccionados para el grabador fueron una tasa de muestreo de 44,1KHz y codificación PCM de 16 bits en un solo canal de audio, obteniendo una calidad de audio de CD.

Para el grabador de audio PCM fue necesario el uso de la clase *AudioRecord* de la plataforma, la cual se encarga de gestionar los recursos de audio de las aplicaciones Java desde la entrada del hardware.

Para esta aplicación se creo la clase *AudioRecorder* que hace uso de la clase *AudioRecord* descrita anteriormente. Esta clase se ejecuta en un hilo independiente para evitar que la interfaz de usuario se bloquee durante la grabación de los ficheros de audio. La figura 4.18 muestra la implementación del método *run()* encargado de inicializar la instancia de la clase *AudioRecord*, configurar los búferes de audio y leer los bytes desde el hardware del dispositivo.

```
44    public void run() {
45        // Wait until recording
46        synchronized(mutex) {
47            while (!this.getRecording()) {
48                try {
49                    mutex.wait();
50                } catch (InterruptedException e) {
51                    e.printStackTrace();
52                }
53            }
54        }
55
56        android.os.Process.setThreadPriority
57            (android.os.Process.THREAD_PRIORITY_URGENT_AUDIO);
58
59        // Allocate Recorder and Start Recording
60        int bufferSize = 2*AudioRecord.getMinBufferSize(this.sampleRate,
61                                                        this.channelConfiguration,
62                                                        this.audioEncoding);
63
64        AudioRecord recorder = new AudioRecord(MediaRecorder.AudioSource.MIC,
65                                               this.sampleRate,
66                                               this.channelConfiguration,
67                                               this.audioEncoding,
68                                               bufferSize);
69
70        int maxSamples = initializeTempBuffer();
71
72        recorder.startRecording();
73
74        samplesRead = 0;
75        int  samplesToRead = bufferSize / 2;
76        int  readResult;
77
78        while (this.getRecording()){
79            // stop recording if we have no more space
80            if(samplesRead + samplesToRead > maxSamples) {
81                this.setRecording(false);
82                continue;
83            }
84
85            readResult = recorder.read(tempBuffer, samplesRead, samplesToRead);
86
87            if (readResult == AudioRecord.ERROR_INVALID_OPERATION ||
88                readResult == AudioRecord.ERROR_BAD_VALUE) {
89                    this.setRecording(false);
90                    continue;
91            }
92            samplesRead += readResult;
93        }
94        recorder.stop();
95    }
```

Figura 4.18. Método *run()* de la clase *AudioRecorder*.

La ejecución de este método es controlada por el usuario mediante el botón de grabar que alterna con cada *click* el valor de la variable booleana *recording.* Por otra parte, se ha fijado un límite de 20 segundos como la máxima duración de un fichero de audio.

Para la facilidad del programador, la clase *AudioRecord* posee un método que calcula el tamaño mínimo recomendable para el búfer de audio del grabador. Para ello, sólo es necesario indicar la frecuencia de muestreo, la cantidad de canales de audio y el tipo de codificación a usar.

Justo después se procede a inicializar la instancia de la clase *AudioRecord.* Para ello es necesario pasar como parámetros la fuente de audio del hardware, en este caso el micrófono del móvil, la frecuencia de muestreo, la cantidad de canales de audio, el tipo de codificación de audio y el tamaño del búfer obtenido anteriormente. Como se mencionó anteriormente, esta instancia se utilizará para leer los bytes de audio desde el hardware.

Uno de los métodos más importantes de la clase *AudioRecord* es *read,* el cual se encarga de leer los datos en *shorts* y guardarlos en un arreglo de tamaño fijo del mismo tipo de variable. Adicionalmente, este método retorna la cantidad de *shorts* que se leyeron permitiendo llevar un conteo de la totalidad de *shorts* leídos hasta el momento y calcular cuántos quedan aún por leer. De esta manera se puede detener la grabación una vez se llega al límite de muestras leídas correspondientes a los 20 segundos de grabación.

Una vez finalizada la grabación se pasan los datos binarios a un fichero *.raw* temporal en la tarjeta SD del dispositivo para que posteriormente agregarle la cabecera WAVE correspondiente. Para ello, se implementó la clase *WaveField* que contiene los métodos necesarios para el cálculo de los diferentes valores de una típica cabecera WAVE[60].

En la figura 4.19 se muestra la inicialización de las constantes y arreglos de bytes, así como el constructor de la clase donde se realizan las llamadas a los diferentes métodos mencionados anteriormente.

```
21    int headerLength = 44;
22    int mAudioFormat = 1;//PCM: Linear quantization
23    int mSampleRate = 8000;
24    int mBitsPerSample = 16;//16 bits
25    int mSubchunk1Size = 16;// for PCM
26    int mNumChannels = 1;//Mono
27    int mSubchunk2Size;
28    int mChunkSize, mNumSamples, mByteRate, mBlockAlign;
29
30    byte[] bChunkSize = new byte[4];
31    byte[] bByteRate = new byte[4];
32    byte[] bSubchunk2ID = new byte[4];
33    byte[] bChunkID = new byte[4];
34    byte[] bSubchunk2Size = new byte[4];
35    byte[] bFormat = new byte[4];
36    byte[] bSubchunk1ID = new byte[4];
37    byte[] bSubchunk1Size = new byte[4];
38    byte[] bSampleRate = new byte[4];
39    byte[] bAudioFormat = new byte[2];
40    byte[] bNumChannels = new byte[2];
41    byte[] bBitsPerSample = new byte[2];
42    byte[] bBlockAlign = new byte[2];
43    byte[] header = new byte[headerLength];
44
45    public WaveField() throws UnsupportedEncodingException{
46        super();
47        this.setChunkID("RIFF");
48        this.setChunkSize(mSubchunk2Size);
49        this.setFormat("WAVE");
50        this.setSubchunk1ID("fmt ");
51        this.setSubchunk1Size(16);
52        this.setAudioFormat(mAudioFormat);
53        this.setNumChannels(mNumChannels);
54        this.setSampleRate(mSampleRate);
55        this.setByteRate(mSampleRate, mNumChannels, mBitsPerSample);
56        this.setBlockAlign(mNumChannels, mBitsPerSample);
57        this.setBitsPerSample(mBitsPerSample);
58        this.setSubchunk2ID("data");
59        this.setSubchunk2Size();
60    }
```

Figura 4.19. Inicialización de variables y constructor de la clase *WaveField.*

Una vez construido el arreglo de bytes con todos los valores de la cabecera, es necesario adjuntarla al resto de los bytes de datos. Para esta tarea se creó un arreglo de bytes del tamaño total que tendrá el fichero WAVE. En la figura 4.20 se puede observar el método *WaveFile()* donde se pasan al arreglo de bytes mencionado los valores de la cabecera WAVE y posteriormente se concatenan los bytes del audio PCM. Una vez obtenido el arreglo de bytes completo se procede a guardarlo en un fichero de nombre aleatorio en una carpeta temporal en la tarjeta SD del dispositivo.

```
171     public void WaveFile() throws IOException{
172
173         //adding header to the wave array
174         byte[] wavearray = new byte[headerLength + mSubchunk2Size];
175         int i = 0;
176         for (i = 0; i < headerLength ; i++){
177             wavearray[i] = header[i];
178         }
179
180         //reading the .raw audio from sdcard and storing in wave array
181         FileInputStream Fin = new FileInputStream(rawfile);
182         Fin.read(wavearray, headerLength, mSubchunk2Size);
183
184         //passing the wavearray to the wavefile in sdcard
185         int count = AddScreen.addeditems.size();
186         Random r = new Random();
187         int filename = r.nextInt(99000);
188         AddScreen.addeditems.get(count-1).file =
189             new File(Environment.getExternalStorageDirectory()+"/temp/"+ filename
    +".wav").toString();
190         FileOutputStream Fout =
191             new FileOutputStream(new File(Environment.getExternalStorageDirectory()
    +"/temp/"+ filename +".wav"));
192         Fout.write(wavearray);
193     }
194 }
```

Figura 4.20. Método *WaveFile()* de la clase *WaveField.*

Una vez se hayan obtenido lo que serán las nuevas partes de la historia, podrá realizar la transformación de audio de las mismas. Para ello, se han implementado esencialmente dos métodos que interactúan con un servicio de transformación de voz ubicado en los servidores remotos del MTG. Estos métodos son *FileCollectionResource* y *TaskCollectionResource* de la clase *TransformTask*. Ambos métodos son ejecutados de manera secuencial para subir y transformar el archivo de audio respectivamente.

En la figura 4.21 se puede observar la implementación de *FileCollectionResource.* Este método se basa en una petición HTTP POST multiparte a la URL http://mtg105.upf.es/api/files, pasando como elementos del cuerpo el archivo binario a subir y el tipo de licencia a utilizar. El valor de esta última es *cc_at_nc* haciendo referencia a un licencia *Creative Commons* de atributo no comercial.

```
27    public void FileCollectionResource(String filepath)throws JSONException,
  IllegalStateException, ClientProtocolException, IOException{
28
29        File file = new File(filepath);
30        String URL = "http://mtg105.upf.es/api/files/";
31        HttpPost POST = new HttpPost(URL);
32
33        FileBody bin = new FileBody(file);
34        StringBody license = new StringBody("cc_at_nc");
35
36        MultipartEntity reqEntity = new MultipartEntity();
37        reqEntity.addPart("license", license);
38        reqEntity.addPart("file", bin);
39        POST.setEntity(reqEntity);
40
41        this.jsonresponse = client.execute(POST).getEntity().getContent();
42        String response = DataRetriever.StreamToString(jsonresponse);
43        this.filekey = new JSONObject(response).getString("_id");
44
45        Log.v("MyActivity 1", this.filekey+"");
46
47    }
```

Figura 4.21. Método *FileCollectionResource* de la clase *TransfromTask*.

En caso que la petición haya sido ejecutada y procesada satisfactoriamente, el servidor responde con un estatus HTTP 201 *Created* y una estructura XML con la referencia, ubicación y clave del recurso creado. Para efectos prácticos sólo la clave del recurso es utilizada por la aplicación.

Una vez subido el archivo de audio es necesario transformarlo. El método *TaskCollectionResource* es el encargado de realizar esta tarea. La figura 4.22 muestra la implementación de dicho método.

Este método también se trata de una petición HTTP POST multiparte, pero en este caso a la URL http://mtg105.upf.es/api/processing/voicetransform/. Los parámetros del cuerpo de la petición son la clave del recurso obtenida con *FileCollectionResource* y el tipo de transformación de voz a aplicar.

```
49    public void TaskCollectionResource(String transform) throws
  IllegalStateException, ClientProtocolException, IOException, JSONException{
50
51        String URL = "http://mtg105.upf.es/api/processing/voicetransform/";
52        StringBody transformation = new StringBody(transform);
53        StringBody key = new StringBody(this.filekey);
54        HttpPost POST = new HttpPost(URL);
55
56        MultipartEntity reqEntity = new MultipartEntity();
57        reqEntity.addPart("preset", transformation);
58        reqEntity.addPart("input", key);
59        POST.setEntity(reqEntity);
60
61        InputStream jsonresponse = client.execute(POST).getEntity().getContent();
62        String response = "{key: " + DataRetriever.StreamToString(jsonresponse) +
  "}";
63        this.taskkey = new JSONObject(response).getString("key");
64
65        Log.v("MyActivity 2", this.taskkey+"");
66
67    }
```

Figura 4.22. Método *TaskCollectionRespurce* de la clase *TransfromTask*.

En caso que la petición se haya solicitado de manera satisfactoria el servidor responde con un estatus HTTP 201 *Created*, la clave del recurso, la fecha de solicitud de la tarea de transformación, la ubicación del recurso en el servidor e indicadores del estado de la transformación.

Para iniciar el flujo de peticiones y respuestas del sistema cliente-servidor para la transformación de voz el usuario deberá presionar el botón de *transform voice* (véase la figura 3.6). Este ejecuta una nueva instancia de la clase *TransformFiles* que extiende *AsyncTask* donde se llevan a cabo los métodos descritos anteriormente.

Como se ha mencionado con anterioridad *AsyncTask* es utilizado para realizar rutinas en hilos de ejecución diferentes al de la GUI para no bloquear los elementos de la vistas.

```
365         protected String doInBackground(Void...voids ) {
366
367             String completed;
368             String successful = "true";
369             TransformTask Transform = new TransformTask();
370
371             //non-UI Thread
372             try{
373                 int i = 0;
374                 for (i = 0 ; i < addeditems.size() ; i++){
375                     if (addeditems.get(i).voice.equals("-")== false
376                             && addeditems.get(i).transformed == false){
377                         //Check for Narrator or for previously transformed parts, if so
    do not Transform
378                         Transform.FileCollectionResource(addeditems.get(i).file+"");
379                         Transform.TaskCollectionResource(addeditems.get(i).voice);
380
381                         //Checks if the transformation is completed
382                         do{
383                             Thread.sleep(500);
384                             completed = Transform.Retriever("completed");
385                         }while(completed.equals("false"));
386
387                         if (Transform.Retriever("successful").equals("true")){
388                             addeditems.get(i).transformed = true;
389                             addeditems.get(i).file = DataRetriever.server +
    DataRetriever.files_url + Transform.Retriever("output_id") +"/conversion/mp3";
390                             addeditems.get(i).key =
    Transform.Retriever("output_id");
391
392                         }else if
     (Transform.Retriever("successful").equals("false")){
393                             //If just one file of the transform batch is
    unsuccessful
394                             //the whole process will be unsuccessful
395                             badtransforms = badtransforms +1;
396                             successful = "false";
397                         }
398                     }
399
400                     if (addeditems.get(i).voice.equals("-")== true
401                             && addeditems.get(i).transformed == false){
402
403                         Transform.FileCollectionResource(addeditems.get(i).file+"");
404
405                         do{
406                             Thread.sleep(500);
407
408                         }while(Transform.filekey == null);
409                         addeditems.get(i).transformed = true;
410                         addeditems.get(i).key = Transform.filekey;
411                     }
412                 }
413             }catch(Exception e){
414                 e.printStackTrace();
415             }
416             return successful;
417         }
```

Figura 4.23. Método *doInBackground()* de la clase *TransformFiles*.

La figura 4.23 muestra el método *doInBackground* de la clase *TransformFiles*. Este método itera en la lista de nuevas partes discriminando si el personaje en cuestión se trata del narrador, que no tiene transformación de voz asociada, o si se trata de una parte que ya ha sido transformada con anterioridad. En caso que ninguna de estas cualidades se cumpla, se procede a llamar a los métodos *FileCollectionResource* y

T*askCollectionResource*. En la línea 382 se puede observar la rutina que monitorea el estado de la trasformación y espera hasta que el estatus de la misma cambie a completado.

Una vez se haya realizado una transformación de manera satisfactoria, la URL del fichero de audio transformado y el identificador del mismo son guardados en los campos correspondientes del objeto que representa a dicha parte para poder ser reproducidos por el usuario. Adicionalmente, se indica una bandera de transformación exitosa para evitar que la misma transformación sea transformada nuevamente. En caso que haya ocurrido un error en alguna de las transformaciones, se incrementa en uno el contador de errores y, una vez concluido el proceso de transformación, se le notifica al usuario que la transformación no ha sido exitosa y el número de partes que no se han podido transformar.

Por otra lado, si el personaje participante se trata del narrador, sólo el método de *FileCollectionResource* será ejecutado ya que no se requiere transformar la voz.

A continuación, en las siguientes figuras se muestran las trazas de las peticiones y respuestas del sistema cliente-servidor del servicio de transformación de voz ya descrito. La figura 4.24 describe la petición POST multi-parte para la subida de un archivo de audio al servicio de transformación de voz.

```
POST /voicetransform/file/ HTTP/1.1
Content-Length: 379159
Content-Type: multipart/form-data; boundary=BGQvssMoEQuzQgN4q1I7BQptNGpPFoJ-1eRnvH-l
Host: mtg105.upf.es
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)
Expect: 100-Continue
Authorization: Basic emZqc2RCSUpEa1dRMHo5dzpad0pCT3BJN25wSEZiUk95

--BGQvssMoEQuzQgN4q1I7BQptNGpPFoJ-1eRnvH-l
Content-Disposition: form-data; name="file"; filename="Wave.wav"
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
                    ----------------------------------------datos binarios----------------------------------------

HTTP/1.1 201 Created
Server: nginx/0.6.32
Date: Wed, 07 Apr 2010 16:10:25 GMT
Content-Type: text/xml
Connection: close

<VoiceTransformFile ref="http://mtg105.upf.es/voicetransform/
file/20100407181025IHgRHTpPvInCronblQVRItIszmxYAlFL" key="20100407181025IHgRHTpPvInCronblQVRItIszmxYAlFL"
file="http://mtg105.upf.es/media/VoiceTransform/20100407181025JpAMJVbwSpDAHVGUYCzwcrwXAxkIviKu.wav"/>
```

Figura 4.24. Traza de la petición/respuesta de *FileCollectionResource*.

La figura 4.25 muestra la petición del inicio de la tarea de la transformación de voz del archivo subido anteriormente en la figura 4.24. Se puede observar que el tipo de transformación utilizado es *HumanToClown1*, o humano a payaso, que procesa la voz a un tono gracioso. Además se puede observar que los estatus de la tarea están todos en *falso*, indicando que aunque la tarea se ha creado aún no se ha iniciado.

```
POST /voicetransform/task/ HTTP/1.1
Content-Length: 418
Content-Type: multipart/form-data; boundary=IJgvKbx8dERMEgSHvMJFEx-3FKKqwcC
Host: mtg105.upf.es
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)
Expect: 100-Continue
Authorization: Basic emZqc2RCSUpEa1dRMHo5dzpad0pCT3BJN25wSEZiUk95

--IJgvKbx8dERMEgSHvMJFEx-3FKKqwcC
Content-Disposition: form-data; name="transform"
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit

HumanToClown1
--IJgvKbx8dERMEgSHvMJFEx-3FKKqwcC
Content-Disposition: form-data; name="input"
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: 8bit

20100407181025IHgRHTpPvInCronblQVRItIszmxYAlFL
--IJgvKbx8dERMEgSHvMJFEx-3FKKqwcC--
HTTP/1.1 201 Created
Server: nginx/0.6.32
Date: Wed, 07 Apr 2010 16:10:30 GMT
Content-Type: text/xml
Connection: close

<VoiceTransformTask key="20100407181030xAsLtIjcCmvsQjhEclmWMWcBmpgxhYPH" ref="http://mtg105.upf.es/voicetransform/
task/20100407181030xAsLtIjcCmvsQjhEclmWMWcBmpgxhYPH" created="2010-04-07 18:10:30.195796" input="http://
mtg105.upf.es/voicetransform/file/20100407181025IHgRHTpPvInCronblQVRItIszmxYAlFL" completed="False"
successful="False" processing="False"/>
```

Figura 4.25. Traza de la petición/respuesta de transformación de voz de *TaskCollectionResource*.

Una vez creada la tarea, la aplicación-cliente deberá periódicamente solicitar el estado de la tarea hasta que se complete. Las figuras 4.26 muestra una de las peticiones HTTP GET y sus respectivas respuestas.

```
GET /voicetransform/task/20100407181030xAsLtIjcCmvsQjhEclmWMWcBmpgxhYPH HTTP/1.1
Host: mtg105.upf.es
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)
Authorization: Basic emZqc2RCSUpEa1dRMHo5dzpad0pCT3BJN25wSEZiUk95

HTTP/1.1 200 OK
Server: nginx/0.6.32
Date: Wed, 07 Apr 2010 16:10:31 GMT
Content-Type: text/xml
Transfer-Encoding: chunked
Connection: close

<VoiceTransformTask key="20100407181030xAsLtIjcCmvsQjhEclmWMWcBmpgxhYPH" ref="http://mtg105.upf.es/
voicetransform/task/20100407181030xAsLtIjcCmvsQjhEclmWMWcBmpgxhYPH" created="2010-04-07 18:10:30"
input="http://mtg105.upf.es/voicetransform/file/20100407181025IHgRHTpPvInCronblQVRItIszmxYAlFL"
completed="False" successful="False" processing="True"/>
```

Figura 4.26. Petición de estatus de transformación de voz. *Procesando.*

La figura 4.27 muestra la traza de una tarea de transformación de voz que se ha completado de manera satisfactoria.

```
GET /voicetransform/task/20100407181030xAsLtIjcCmvsQjhEclmWMWcBmpgxhYPH HTTP/1.1
Host: mtg105.upf.es
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)
Authorization: Basic emZqc2RCSUpEa1dRMHo5dzpad0pCT3BJN25wSEZiUk95

HTTP/1.1 200 OK
Server: nginx/0.6.32
Date: Wed, 07 Apr 2010 16:10:51 GMT
Content-Type: text/xml
Transfer-Encoding: chunked
Connection: close

1be
<VoiceTransformTask key="20100407181030xAsLtIjcCmvsQjhEclmWMWcBmpgxhYPH" ref="http://mtg105.upf.es/
voicetransform/task/20100407181030xAsLtIjcCmvsQjhEclmWMWcBmpgxhYPH" created="2010-04-07 18:10:30"
input="http://mtg105.upf.es/voicetransform/file/20100407181025IHgRHTpPvInCronblQVRItIszmxYAlFL" output="http://
mtg105.upf.es/voicetransform/file/20100407181051OLKIfbWBPgjKXtAUieLskZGcJLSQqzrX" completed="True"
successful="True" processing="False"/>
```

Figura 4.27. Petición de estatus de transformación de voz. *Completada.*

Una vez que las partes hayan sido transformadas satisfactoriamente, el usuario podrá proceder a subirlas al servidor de la aplicación para hacer pública su colaboración y dándole continuidad a la historia. Para ello, el usuario sólo deberá presionar el botón de *upload*, o subida, ubicado en la esquina inferior derecha (véase la figura 3.6) .

La figura 4.28 muestra el método que ejecuta la rutina del botón mencionado anteriormente. En primer lugar se crea una instancia de la clase *PartUploader*, que aloja los métodos *contentbuilder()* y *upload(),* encargados de crear el objeto en formato JSON con la información de las partes a subir y la ejecución de la petición HTTP que enviará dicha información al servidor respectivamente. Una vez finalizada la petición se muestra por pantalla a través de un mensaje *Toast* el estatus de la respuesta del servidor. En caso que esta sea positiva, se indicará un estatus *Created*, o creado, y se procederá a borrar los ficheros de audio temporales y se limpiará la lista de partes pendientes por subir.

```
 98          ImageButton UploadParts = (ImageButton) findViewById(R.id.UploadParts);
 99          UploadParts.setOnClickListener(new ImageButton.OnClickListener(){
100
101              PartUploader Uploader = new PartUploader();
102
103              @Override
104              public void onClick(View arg0) {
105
106                  try {
107                      Uploader.contentbuilder();
108                      Uploader.upload();
109
110                      Toast success = new Toast(AddScreen.this);
111                      success = Toast.makeText(AddScreen.this, PartUploader.status,
    5000);
112                      success.show();
113
114                      String status = "Created";
115
116                      if ((PartUploader.status).equalsIgnoreCase(status)==true){
117                          removeDir();
118                          addeditems.clear();
119                          ListUpdater();
120                      }
121
122                  } catch (Exception e) {
123                      e.printStackTrace();
124                  }
125              }
126          });
```

Figura 4.28. Implementación del botón de subida de las nuevas partes.

Para mayor entendimiento de la comunicación del modelo cliente-servidor de la aplicación para la tarea de la subida de las nuevas partes de la historia, se explicará la implementación de los métodos *contentbuilder()* y *upload().* El primero de ellos, mostrado en la figura 4.29, utiliza la clase *JSONStringer* de la API de la plataforma *Android* para la creación y recopilación de información en formato JSON. Se crea un arreglo de objetos JSON, donde cada uno representa una nueva parte a subir. Los objetos JSON deben tener cuatro campos; la clave obtenida durante el proceso de transformación que identifica al recurso en la base de datos, el nombre del personaje y la ciudad y país de ubicación del usuario al momento de la ejecución de este método.

```
39    public void contentbuilder() throws JSONException{
40
41        JSONStringer parts;
42        user = Login.user_login;
43        lock = PartScreen.lock;
44
45        parts = new JSONStringer().array();
46
47        int i = 0;
48        for (i = 0 ; i < AddScreen.addeditems.size(); i++){
49
50        parts.object()
51                    .key("key").value(AddScreen.addeditems.get(i).key)
52
              .key("character").value(AddScreen.addeditems.get(i).character_name)
53             .endObject();
54        }
55        parts.endArray();
56        this.parts = parts.toString();
57    }
58 }
59
```

Figura 4.29. Método *contentbuilder()* de la clase *PartUploader*.

Por otra parte, el método *upload()*, mostrado en la figura 4.30, se encarga de construir y ejecutar la petición HTTP POST multiparte a la URL *http://mtg105.upf.es/api/tales/stories/[stroy_id]/parts*. Esta petición contiene el nombre del usuario, el documento JSON construido en el método *contentbuilder()* y el *lock*, o cerradura, obtenida al iniciar la actividad con el fin de evitar la concurrencia de múltiples usuarios al momento de modificar las historias.

```
23    public void upload() throws ClientProtocolException, IOException{
24
25        String URL = DataRetriever.server + DataRetriever.story_url +
  StoryScreen.story + "/parts";
26        StringBody user = new StringBody(this.user);
27        StringBody lock = new StringBody(this.lock);
28        StringBody parts = new StringBody(this.parts);
29        HttpPost POST = new HttpPost(URL);
30        MultipartEntity reqEntity = new MultipartEntity();
31        reqEntity.addPart("user", user);
32        reqEntity.addPart("parts", parts);
33        reqEntity.addPart("lock", lock);
34        POST.setEntity(reqEntity);
35        InputStream response = client.execute(POST).getEntity().getContent();
36        status = DataRetriever.StreamToString(response);
37    }
```

Figura 4.30. Método *upload()* de la clase *PartUploader.*

Como se ha mencionado anteriormente a lo largo de estas memorias, este proyecto sólo busca presentar una versión de demostración de Tales y no debe ser pensado como un producto terminado y listo para su publicación. Por esta razón, esta versión no posee

herramientas de monitoreo de red, con lo cual no puede responder de manera ideal en caso de errores de conexión.

Tanto del lado del cliente como del servidor, el sistema de la aplicación no soporta el reestablecimiento de las transferencias de los ficheros de audio de las partes así como su transformación. En caso de ocurrir una desconexión el usuario deberá iniciar desde el comienzo cualquier transferencia incompleta. Si los problemas de conexión ocurren durante la solicitud de la metadata así como las partes de la historia, no se puede esperar un comportamiento apropiado de la aplicación.

Por otra parte, la información de localización también se verá afectada por problemas en la red. En caso de una desconexión el gestor de localización no podrá actualizar la información de localización, pero sí mantendrá la ubicación obtenida la última vez que se realizó la actualización, obteniendo un resultado aceptable en la mayoría de los casos.

Capítulo 5
Conclusiones

Este capítulo, el último de este documento, tratará en primer lugar los objetivos alcanzados durante el desarrollo de la aplicación. También se mencionarán algunas mejoras importantes y otros detalles implementación y usabilidad pendientes por realizar, que harán de Tales una aplicación apta para entrar en el competitivo mercado de las aplicaciones móviles. Se tratarán temas relacionados con posibles futuras líneas de investigación y desarrollo dentro de este creciente campo. También se mencionarán los aspectos más importantes de la plataforma Android que se trataron durante el desarrollo y documentación del proyecto. Finalmente se comentarán las principales virtudes de la documentación en LaTeX.

5.1 Objetivos Alcanzados

Este proyecto ha conseguido alcanzar todos los objetivos que se propusieron al iniciar la idealización y creación de esta aplicación.

En primer lugar, se consiguió satisfactoriamente desarrollar la aplicación haciendo uso de las herramientas ofrecidas por la plataforma Android. Se logró obtener una versión alfa de la aplicación, donde se encuentran operativas sus funcionalidades más importantes y muestra el potencia de esta plataforma. Esta versión permite a los integrantes de Tales

participar de manera pasiva y activa en historias ya existentes, dándole un carácter móvil a esta comunidad de cuenta cuentos.

Adicionalmente, se obtuvieron grandes conocimientos en el manejo de las tecnologías relacionadas con un proyecto de este tipo. Se aprendieron conceptos importantes para el desarrollo de software dentro de las limitaciones típicas de un dispositivo móvil. También se obtuvieron conocimientos avanzados en el uso del lenguaje de programación y un buen manejo de las diferentes APIs que hacen posible desarrollar aplicaciones bajo la plataforma Android.

Se comprendieron los elementos más importantes a tener en cuenta al momento de diseñar e implementar una GUI que permita a los usuarios navegar a través de la aplicación de una manera fluida e intuitiva. De hecho, es esta una de las principales virtudes de la versión alpha de esta aplicación; una interfaz gráfica atractiva y audiovisual.

Por otra parte, a medida que la aplicación fue tomando forma, se implementaron funciones adicionales al diseño original. Esto sucedió ya que se notaron algunas deficiencias y posibles mejoras que no representaban grandes inconvenientes desde el punto de vista de implementación, pero otorgaban considerables mejoras en la usabilidad general de la aplicación.

Finalmente, se hizo un simple estudio económico con los costes de implementación de una aplicación de este tipo. Esto, junto con la investigación pertinente del negocio de las aplicaciones móviles, permitió llegar a conclusiones importantes respecto a la rentabilidad económica del desarrollo dentro de estas tecnologías. Hay que notar que, aunque el negocio de las aplicaciones móviles puede llegar a ser lucrativo, no se pueden subestimar esfuerzo de desarrollo necesario ni la alta competitividad del mercado.

5.2 Mejoras

El haber obtenido una versión alfa de la aplicación deja un amplio terreno para planear, diseñar e implementar nuevas funcionalidades. Adicionalmente, se pueden mejorar aquellas ya existentes haciendo uso de versiones más recientes de la plataforma Android y las APIs que estas ofrecen.

Una de las principales cosas que la aplicación móvil de Tales buscará ofrecer próximamente es independizarse de la página Web en la medida de lo posible, es decir, se buscará ofrecer desde el dispositivo móvil funcionalidades que actualmente sólo encuentran disponibles desde la plataforma Web. Una de las funciones más importantes sería la posibilidad de crear historias y personajes desde el dispositivo móvil. Esto se obvió en la versión actual, ya que se pensó que sería un poco tedioso para el usuario la labor de escribir mucho texto y añadir los recursos gráficos necesarios para poder crear la historia y los personajes. No obstante a medida que los teclados táctiles y exploradores Web de dispositivos móviles vayan mejorando, esta labor será cada vez más sencilla.

El ámbito gráfico es un aspecto clave en cualquier aplicación móvil como Tales, ya que la GUI afecta de manera importante cómo el usuario percibe la aplicación y cómo interactúa con ella. Aunque la función, la respuesta y la robustez es lo que hace que usuario permanezca como usuario, las aplicaciones siempre entran primero por los ojos y luego por los dedos.

5.2.1 Tales y Facebook

Tal como se ha mencionado a lo largo de esta memoria, la aplicación de Tales busca dar soporte móvil a una comunidad virtual de cuenta cuentos, siendo sumamente importante tener una significativa base usuarios para alcanzar un verdadero éxito. Teniendo esto en cuenta, es fundamental la viralidad de la aplicación y llegar a la mayor cantidad de personas posibles para que compartan un espacio virtual determinado. Para ello, se tiene pensada una mejora que resolverá este y más aspectos en la aplicación, representando un punto de inflexión en el futuro próximo de Tales; integrarla con Facebook.

Facebook, con más de 500 millones de usuarios en todo el mundo, es la principal red social que jamás haya existido. Sus usuarios permanecen conectados más 700 billones de minutos al mes, creando más de 30 billones de piezas de contenido. Adicionalmente, Facebook es más que una red social, es una plataforma que permite a desarrolladores crear aplicaciones e integrar algunas de sus funcionalidades en otras Webs. Facebook también otorga movilidad, con más de 250 millones de usuarios que acceden desde sus diferentes terminales móviles[29].

Las APIs de Facebook para Android ofrecerán importantes funcionalidades para la aplicación de Tales. Entre estas destacan las siguientes[28]:

- Una robusta y segura herramienta que permitirá a los usuarios de Tales ingresar a la aplicación usando el sistema de autenticación y cuenta de Facebook. Esto facilita en gran medida las labores de *backoffice* que representa implementar protocolos de autenticación y gestionar las diferentes cuentas de usuario.
- Obtener la lista de amigos de la cuenta de Facebook del usuario lo cual le dará la característica de comunidad virtual a Tales, permitiendo invitar a otros usuarios a participar en las historias y compartir las ya existentes.
- Hacer publicaciones en el *Wall* de Facebook lo cual permitirá a un gran número de usuarios notar la actividad de los usuarios de Tales y dar a conocer la aplicación. Esto es un aspecto fundamental para la viralidad y crecimiento de la base de usuarios de la aplicación.
- Implementar motores para publicidad y créditos, permitiendo monetizar la aplicación y encontrar rentabilidad económica en la labor de desarrollo. Adicionalmente, ya que las APIs de Facebook te permiten obtener información acerca del usuario, es posible llevar a cabo una publicidad más inteligente y dirigirla a perfiles específicos.

Estos puntos darán a Tales las características necesarias para posicionarse como una aplicación de calidad, novedosa y gran valor agregado.

Queda claro que el desarrollo de software es una tarea cíclica donde siempre habrán aspectos que retomar, rediseñar, probar y depurar. Se podría decir que, mientras las APIs, los dispositivos móviles y el mercado de aplicaciones móviles evolucionen, aplicaciones como Tales siempre dejarán lugar para mejorar y ofrecer nuevas funcionalidades. Por otra parte, la retroalimentación es una de las principales formas de encontrar mejoras. Por ello, a medida que la base de usuarios crezca y la aplicación perdure activamente en el mercado, se obtendrá información clave acerca de cuál dirección deberá tomar la aplicación en el futuro.

5.3 Futuras Líneas de Investigación

Antes de escribir la primera línea de código, Tales se pensó como un espacio virtual donde personas podrían colaborar en la creación de historias y compartirlas con los diferentes miembros de esta comunidad de cuenta cuentos. Sin embargo, a pesar de tener esta finalidad específica, son los usuarios y el tiempo quienes en verdad dictan la manera en la cual la aplicación es utilizada. Esto, aunque pueda parecer una imprecisión, permite abstraer los valores tecnológicos de la aplicación y trasladarlos a diferentes líneas de investigación. La creación de contenido audiovisual y la posibilidad de compartirlo, la colaboración creativa y geoposicionamiento entre usuarios, todo dentro de un contexto libre y móvil, es lo que hace de Tales una herramienta útil más allá de su finalidad lúdica.

En el ámbito de la enseñanza se podría encontrar gran utilidad a una aplicación como Tales. Maestros podrían, por ejemplo, preparar lecciones donde los propios personajes históricos relaten los sucesos que los hicieron importantes. También se podrían representar relatos de alguna obra literaria donde quizás los mismos alumnos remotamente desde sus dispositivos móviles colaboren en su creación. También se podría utilizar Tales en la enseñanza de lenguas extrajeras, permitiendo a los usuarios practicar la fonética y aprender nuevo vocabulario. Estas son sólo algunas de las aplicaciones que Tales puede ofrecer en el campo de la enseñanza y con la ayuda de expertos en pedagogía y aprendizaje se podrían encontrar muchas más.

Tales, en esencia, posee las principales funcionalidades necesarias para crear un servicio de mensajería de voz, permitiendo intercambiar notas de audio y tener conversaciones en diferido entre dos o más usuarios y, si se toma en cuenta la propuesta integración de Tales con Facebook, se obtendría un servicio adicional que la red social aún no posee.

También se podría entrar en el mundo de la literatura o cómics, trasladando relatos a la plataforma de Tales y permitiendo a los usuarios escuchar y ver en cierta medida como los personajes se desenvuelven a lo largo de la historia. De esta manera se incurriría en el negocio de los audio-libros, pero dando un paso más allá gracias a los servicios de transformación de voz y capacidad visual de Tales.

Estas son sólo algunas de las posibles líneas que la aplicación de Tales podría seguir, dando espacio para desarrollar nuevos proyectos de investigación y ofrecer nuevos servicios con un valor tecnológico adicional y novedoso.

5.4 Plataforma Android

A lo largo del desarrollo de este proyecto, desde el proceso de investigación hasta la escritura de estas memorias, se ha podido llegar a numerosas conclusiones respecto a la plataforma Android y lo esta busca alcanzar.

En primer lugar, el hecho que se haya escogido Java como su lenguaje de programación permite que un gran número de programadores que ya conocían este lenguaje puedan desarrollar aplicaciones para la plataforma. Además, la amplia documentación hacen la migración de J2ME a Android relativamente fácil con sus numerosos tutoriales y códigos de muestra. No obstante, como fue el caso de este proyecto, el uso de Java permitió desarrollar aplicaciones para Android sólo con previos conocimientos básicos en programación orientada a objetos.

Otro aspecto importante a destacar son las poderosas APIs que se ofrecen, desde la facilidad para crear GUIs hasta los controladores de los diferentes elementos de hardware del dispositivo, permitiendo crear aplicaciones completas de una manera transparente para el desarrollador. Además de esto es posible importar librerías externas e integrarlas sin grandes complicaciones en el código, aunque estas no hayan sido creadas para la plataforma. También hay que tener en cuenta el concepto ecosistema de la plataforma otorgando una gran integración de funciones de diferentes aplicaciones.

El equipo de desarrolladores de Android trabaja constantemente en nuevas versiones de la plataforma, resolviendo los problemas existentes y sobrellevando las deficiencias del sistema. No obstante, esto trae uno de los principales inconvenientes de Android, la fragmentación. El hecho que hayan múltiples versiones del mismo sistema operativo conviviendo simultáneamente complica la tarea de desarrollar aplicaciones. El desarrollador tiene la difícil tarea de sacar el máximo provecho de lo que la plataforma ofrece, pero procurar dejar atrás a la menor cantidad de usuarios con versiones más antiguas.

Adicionalmente, al ser una plataforma libre que no está sujeta a un solo fabricante de dispositivos móviles, cada uno de ellos busca diferenciarse del resto montando software propietario sobre el sistema operativo. Por otra parte, los diferentes Carriers preinstalan diferentes aplicaciones en las ROM con el mismo objetivo. Si agregamos esto al problema de fragmentación antes mencionado, la diversidad y libertad de la plataforma complica mucho las cosas.

Otra aspecto a importante en referencia a la diversidad del mundo Android son sus múltiples mercados de aplicaciones. A diferencia del centralizado *AppStore*[13] para iPhone, en Android existen varios mercados alternativos al *Android Market*[9] recomendado por Google. Diversos mercados independientes o de alianzas entre Carriers y proveedores de servicios ofrencen a los desarrolladores otros lugares donde publicar sus aplicaciones. Lo que podría parecer una ventaja en realidad hace la labor de desarrolladores y usuarios más difícil dispersando el foco de las ventas. Además de esto, los mercados trabajan de manera muy similar cobrando lo misma comisión por descarga y ofreciendo prácticamente el mismo servicio.

Dentro del contexto comercial detrás de la plataforma Android, es clave comentar que a pesar de ser el sistema operativo con la mayor en el mercado, las ventas de aplicaciones dejan mucho que desear, obteniendo apenas el 4.7% de las ganancias por venta de aplicaciones y posicionándose en el cuarto lugar. Por otro lado, el AppStore de Apple obtuvo el 82.7% posicionándose como el indiscutible ganador, seguido por el AppWorld de BlackBerry[18] y el Ovi Store de Nokia[52] con 7.7% y 4.9% respectivamente[39]. Es difícil apuntar la causa de estas bajas ventas, pero parece ser que el entorno libre y Open Source hace que los usuarios perciban a Android como un entorno donde todo viene gratis.

Preguntas que muchos desarrolladores y personas involucradas en el mundo de las aplicaciones se hacen frecuentemente son *¿Cuál plataforma es mejor, Android o iOS? ¿Llegará Android a desplazar a iPhone?*. Después de todo el proceso de investigación y desarrollo invertidos en este proyecto las respuestas continúan siendo inconclusas.

En el apartado 2.2.6 de estas memorias se comentaron algunas diferencias entre Android y iOS. Sin embargo, aunque esta comparación es válida y está basada en hechos, no se

puede decir cual plataforma es mejor. Simplemente son diferentes y buscan cosas diferentes. Desde su lanzamiento, Apple a conseguido con el iPhone cambiar el paradigma en mundo de la tecnología móvil y la percepción de sus dispositivos logrando crear un sistema que es la envidia de la competencia.

Por otra parte, Android ha conseguido crear una plataforma libre sobrellevando de una manera positiva los obstáculos que conlleva la diversidad de dispositivos y fabricantes en los cuales trabaja. En tan solo cuatro años Android ha logrado posicionarse como la principal plataforma móvil en el mundo, ayudando a la presencia ubicua de los Smartphones en el mercado y proliferando el cambio de paradigma del iPhone de manera masiva.

Bibliografía

[1] Accuhash. What id MD5? 9 Febrero 2006. http://www.accuhash.com/what-is-md5.html. [Online; accessed 12-October-2010].

[2] Android Developers. 21 Septiembre 2010. User Interface. http://developer.android.com/guide/topics/ui/index.html. [Online; accessed 29-September-2010].

[3] Android Developers. 25 Septiembre 2010. Intent. http://developer.android.com/reference/android/content/Intent.html. [Online; accessed 27-September-2010].

[4] Android Developers. Application Fundamentals. 26 Spetiembre 2010. http://developer.android.com/guide/topics/fundamentals.html. [Online; accessed 27-September-2010]

[5] Android Developers. Designing for Performance. 17 Agosto 2010. http://developer.android.com/guide/practices/design/performance.html. [Online; accessed 8-October-2010].

[6] Android Developers. Designing for Responsivness. 15 Septiembre 2010. http://developer.android.com/guide/practices/design/responsiveness.html. [Online; accessed 9-October-2010].

[7] Android Developers. What is Android?. 22 Spetiembre 2010. http://developer.android.com/guide/basics/what-is-android.html. [Online; accessed 23-September-2010].

[8] Android Feeder. Android Market. http://androidfeeder.com/. [Online; accessed 11-October-2010].

[9] Android Market. https://market.android.com/. [Online; accessed 10-June-2011].

[10] Apache Projects. Apache http Server. 19 Octubre 2010. http://projects.apache.org/projects/http_server.html. [Online; accessed 22-October-2010].

[11] Apache. http://www.apache.org/. [Online; accessed 22-October-2010].

[12] Apple. iPhone. http://www.apple.com/iphone/apps-for-iphone/#heroOverview. [Online; accessed 11-October-2010].

[13] Apple.com. Apps for iPhone. http://www.apple.com/es/iphone/apps-for-iphone/. [Online; accessed 10-June-2011].

[14] BAILOR, Coreen. Mobile Is Set to Get Hotter. Destination CRM [en línea]. 30 de Mayo de 2007. http://www.destinationcrm.com/Articles/CRM-News/Daily-News/Mobile-Is-Set-to-Get-Hotter-47733.aspx.. [Online; accessed 11-Septiembre-2010].

[15] Beavis, Gareth. Revealed: Android Honeycomb next up from Google. 19 August 2010. http://www.techradar.com/news/phone-and-communications/mobile-phones/revealed-android-honeycomb-next-up-from-google-711132. [Online; accessed 23-September-2010].

[16] BENEDETTI, Winda. Pigs outraged as Angry Birds sells 12 million copies. 8 Diciembre 2010. http://technolog.msnbc.msn.com/_news/2010/12/08/5612223-pigs-outraged-as-angry-birds-sells-12-million-copies. [Online; accessed 28-January-2011].

[17] BERNERS-LEE, T. Uniform Resouce Identifiers. Agosto 1998. http://www.ietf.org/rfc/rfc2396.txt. [Online; accessed 28-October-2010].

[18] Blackberry.com. App World. http://us.blackberry.com/apps-software/appworld/. [Online; accessed 10-June-2011].

[19] BLAKE, Vikki. Plants vs. Zombies is Most Successful App Launch Ever. 25 Febrero 2010. http://www.ggsgamer.com/2010/02/25/plants-vs-zombies-is-most-successful-app-launch-ever/. [Online; accessed 28-January-2011].

[20] BORT, Dave. Android is now available as open source. 21 October 2008. Android Open Source Project. [Accessed 23-September-2010].

[21] Business Week Online. 23 Julio 2005. http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm. [Online; accessed 22-September-2010].

[22] Canalys.com. Google's Android becomes the World's leading Smartphone Platform. 31 Enero 2011. http://www.canalys.com/pr/2011/r2011013.html. [Online; accessed 10-June-2011].

[23] CASAS MANGAS, David. Desenvolupamnet d'aplicacions lliures per a telefons mobils usant la plataforma Android. 22 Enero 2009.

[24] CERF, Vint y KAHN, Robert. Transmission Control Protocol. Septiembre 1981. http://www.faqs.org/rfcs/rfc793.html. [Online; accessed 22-October-2010].

[25] DIERKS, T. The TLS Protocol. Enero 1999. http://www.ietf.org/rfc/rfc2246.txt. [Online; accessed 28-October-2010].

[26] Eclipse. http://www.eclipse.org/. [Online; accessed 12-October-2010].

[27] ESCALLIER, Paul. 10 Things Android Does Better Than iPhone. 6 Marzo 2010. http://www.maximumpc.com/article/features/10_things_android_does_better_iphone?page=0,0. [Online; accessed 11-October-2010].

[28] Facebook Developers. Advanced Topics. 9 Merzo 2011. http://developers.facebook.com/docs/advancedtopics/. [Online; accessed 15-March-2011].

[29] Facebook. Statistics. http://www.facebook.com/press/info.php?statistics. [Online; accessed 15-March-2011].

[30] Farmville. http://www.farmville.com/. [Online; accessed 23-January-2011].

[31] FIELDING, R. Hypertext Transfer Protocol. Junio 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.html. [Online; accessed 28-October-2010].

[32] FLEICHMAN, E. WAVE and AVI Codec Registries. Junio 1998. http://tools.ietf.org/html/rfc2361. [Online; accessed 28-October-2010].

[33] Flickr. http://www.flickr.com/. [Online; accessed 11-October-2010].

[34] Gartner Newsroom. Gartner Says Consumers Will Spend $6.2 Billion in Mobile Application Stores in 2010. 18 Enero 2010. http://www.gartner.com/it/page.jsp?id=1282413. [Online; accessed 28-January-2011].

[35] Gartner, Inc. 19 Mayo 2010. http://www.gartner.com/it/page.jsp?id=1372013. [Online; accessed 10-September-2010].

[36] Google Blog. Celebrating Android. 23 Junio 2010. http://googleblog.blogspot.com/2010/06/celebrating-android.html. [Online; accessed 11-October-2010].

[37] Google. Gmail. http://mail.google.com/. [Online; accessed 11-October-2010].

[38] HABBO. http://www.habbo.es/. [Online; accessed 23-January-2011].

[39] HALSAM, Oliver. Apple’s iOS Dominates Mobile Software Market Sales. 21 Febrero 2010. http://touchreviews.net/ios-dominates-mobile-software-market-sales/ [Online; accessed 19-June-2011].

[40] HASHIMI, Sayed. The History of Google Android. 5 Marzo 2010. http://www.ctoedge.com/content/history-google-android. [Online; accessed 23-September-2010].

[41] Heinencreative.com. How to Profit From Mobile Apps?. 27 Abril 2010. http://heinencreative.com/archives/articles/how-to-profit-from-mobile-apps/. [Online; accessed 23-January-2011].

[42] IBM. Standards and Specs: The Interchange File Format. 13 Junio 2006http://www.ibm.com/developerworks/power/library/pa-spec16/. [Online; accessed 2-November-2010].

[43] Jet Audio. http://www.jetaudio.com/. [Online; accessed 12-October-2010].

[44] JSON.org. Introducing JSON. http://www.json.org/. [Online; accessed 22-October-2010].

[45] Khronos. OpenGL 1.3 Especification. 24 Abril 2008. http://www.khronos.org/registry/gles/extensions/OES/OES_compressed_ETC1_RGB8_texture.txt. [Online; accessed 12-October-2010].

[46] LiMo Fundation. http://www.limofoundation.org/. [Online; accessed 22-September-2010].

[47] Music Technology Group. http://mtg.upf.edu/project/music3.0. [Online; accessed 22-September-2010].

[48] National Marine Electronics Association. NMEA 0183 Standard. http://www.nmea.org/content/nmea_standards/nmea_083_v_400.asp.[Online; accessed 22-October-2010].

[49] NORDYKE, Kimberly. Social gaming market to pass $1 billion in 2011. Enero 14 2011. http://www.reuters.com/article/idUSTRE70C60L20110114. [Online; accessed 23-January-2011].

[50] Open Handset Alliance. 5 Noviembre 2007. http://www.openhandsetalliance.com/press_110507.html. [Online; accessed: 22-September-2010].

[51] OpenMoko. 17 Septiembre 2010. http://wiki.openmoko.org/wiki/Main_Page. [Online; accessed 22-September-2010].

[52] OVI Store. http://store.ovi.com/. [Online; accessed 10-June-2011].

[53] POSTEL, J. User Datagram Protocol. 28 Agosto 1980. http://www.faqs.org/rfcs/rfc768.html. [Online; accessed 22-October-2010].

[54] PRATT, Justine. Earning a Living as an Independent Mobile Software Developer. 10 Enero 2010. http://www.creativealgorithms.com/blog/content/earning-living-independent-mobile-software-developer. [Online; accessed 23-January-2011].

[55] QEMU Open Source Processor Emulator. About. http://wiki.qemu.org/Main_Page. [Online; accessed 22-October-2010].

[56] RESCORLA, E. The Secire Hypertext Transfer. Agosto 1999. http://www.rfc-editor.org/rfc/rfc2660.txt. [Online; accessed 28-October-2010].

[57] Rovio.com. Angry Birds. http://www.rovio.com/index.php?page=angry-birds [Online; accessed 28-January-2011].

[58] SHI, Yunhe. Virtual Machine Showdown: Stack Versus Registers. 6 Junio 2005. [Online; accessed 23-September-2010].

[59] SWETLAND, Brian. Some clarification on “the Android Kernel”. 7 Febrero 2010. [Accessed 23-September-2010].

[60] Telnet.org. http://www.telnet.org/htm/dev.htm. [Online; accessed 22-October-2010].

[61] The Nielsen Company. 26 Marzo 2010. http://blog.nielsen.com/nielsenwire/consumer/ smartphones-to-overtake-feature-phones-in-u-s-by-2011/. [Online; accessed 10-September-2010].

[62] The NPD Group. Motorola, HTC Drive Android to Smartphone OS Lead in the U.S. 10 Mayo 2010. http://news.yahoo.com/s/prweb/20100804/bs_prweb/prweb4350164. [Online; accessed 11-June-2010].

[63] Wikipedia. Advanced RISC Machines. 2 Septiembre 2011. http://es.wikipedia.org/wiki/ARM. [Online; accessed 28-October-2010].

[64] Wikipedia. Application Programming Interface. Mayo 2010. http://en.wikipedia.org/wiki/Application_ programming_interface. [Online; accessed 17-Septembre-2010].

[65] Wikipedia. Cliente-Servidor. 2 Junio 2011. http://es.wikipedia.org/wiki/Cliente-servidor. [Online; accessed 10-June-2011].

[66] Wikipedia. Handle. 12 Marzo 2010. http://es.wikipedia.org/wiki/Handle. [Online; accessed 12-October-2010].

[67] Wikipedia. Hilo de Ejecución. 17 Julio 2010. http://es.wikipedia.org/wiki/Hilo_de_ejecuci%C3%B3n. [Online; accessed 12-October-2010].

[68] Wikipedia. HTTP Cookie. 22 Octubre 2010. http://en.wikipedia.org/wiki/HTTP_cookie. [Online; accessed 28-October-2010].

[69] Wikipedia. Interface Description Language. 2 Spetiembre 2010. http://es.wikipedia.org/wiki/Interface_description_language. [Online; accessed 12-October-2010].

[70] Wikipedia. Memory Management Unit. 21 Octubre 2010. http://en.wikipedia.org/wiki/Memory_management_unit. [Online; accessed 28-October-2010].

[71] Wikipedia. Multipurpouse Internet Mail Extensions. 16 Octubre 2010. http://es.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions. [Online; accessed 22-October-2010].

[72] Wikipedia. Resource Interchage File Format. 12 Octunre 2010. http://es.wikipedia.org/wiki/Resource_Interchange_File_Format. [Online; accessed 28-October-2010].

[73] Wikipedia. SQLite. 28 Agosto 2010. http://es.wikipedia.org/wiki/SQLite. [Online; accessed 23-September-2010].

[74] Wikipedia. Transmission Control Protocol. 4 Octubre 2010. http://es.wikipedia.org/wiki/Transmission_Control_Protocol. [Online; accessed 28-October-2010].

[75] WILSON, Scott. WAVE PCM Soundfile Format. 20 Enero 203. https://ccrma.stanford.edu/courses/422/projects/WaveFormat/. [Online; accessed 2-November-2010].

[76] World Wide Web Consortium. Extensible Markup Language. 23 Junio 2010. http://www.w3.org/XML/. [Online; accessed 29-September-2010].

[77] XMPP Standards Foundation. http://xmpp.org/. [Online; accessed 12-October-2010].

Agradecimientos

A mis padres por todo su cariño y apoyo y por ser un ejemplo constante a seguir. A Grace por su compresión y sus consejos. A mis amigos y compañeros por compartir largas horas de estudio y buenos recuerdos. Al MTG por la oportunidad y ayuda que me dieron. A la comunidad de desarrolladores de Android por hacer lo que les gusta.

A todos, gracias

Anexo A
Planificación y Costes

Este proyecto nace como una iniciativa propia y de algunos investigadores del *Music Technology Group* con la finalidad de desarrollar tecnología dentro de la filosofía del *Music 3.0*. Inicialmente no se tenía una idea concreta para el enfoque del proyecto pero, después de diversas discusiones, se pensó en relacionarlo con aplicaciones móviles. De esta manera se deseó crear una aplicación móvil que integrara tecnologías propias del departamento a la vez que pudiera ser utilizada por otras personas, aportando de esta manera valor tecnológico a una creciente red de colaboración. En este capítulo se presentará la distribución de las tareas realizadas y el tiempo empleado en cada una de ellas. Adicionalmente, se mostrará el costo aproximado para el desarrollo de un proyecto de este tipo.

A.I Planificación de tareas

En la tabla A.1 se muestra la planificación de las tareas necesarias para el desarrollo de la aplicación y una estimación de las horas empleadas en cada una de ellas. Primero se realizó una fase de tormenta de ideas y desarrollo creativo para encontrar un proyecto que se adaptara las exigencias del *Music Technology Group,* de igual manera que cumpliera las expectativas académicas de un Proyecto Final de Carrera de la Escuela Técnica Superior de Ingeniería de Telecomunicaciones de Barcelona de la Universidad

Politécnica de Cataluña. Una vez adoptada la idea de desarrollar una aplicación móvil, se procedieron a estudiar las diferentes plataformas disponibles y otros aspectos tecnológicos relacionados con este tipo de desarrollo de software. Por las razones descritas en capítulos anteriores, se escogió Android como plataforma de desarrollo.

Posteriormente, se realizó un estudio exhaustivo de la plataforma a través de la documentación oficial disponible y se procedieron a implementar una serie de demos para la familiarización con algunos conceptos básicos de implementación bajo esta la plataforma.

Tareas	Tiempo Invertido (horas)
Fase creativa	**30**
Estudio de las posibles plataformas	**12**
Estudio de la plataforma seleccionada	**80**
Tutorialización y demos	**40**
Diseño de la aplicación	**20**
Implementación	**420**
Captura y registro de información de historias	32
Visualizador de la historia	80
Grabador de audio PCM	88
Cabecera WAVE	32
Captura y registro de segmentos de historia	8
Visualizador de segmentos de historia	48
Reproductor de archivos de audio	28
Geolocalizador y etiquetas de posición	6
Herramientas de captura de audio	24
Visualizador de nuevos segmentos de historia	36
Transformador de voz y subida	8
Acceso y registro de usuarios	6
Cámara fotográfica	10
Pruebas	**14**
Depuración y mejoras	**22**
Documentación	**176**
Total	**814**

Tabla A.1. Planificación del coste temporal de tareas.

Teniendo en cuenta las capacidades y limitaciones tecnológicas de la plataforma Android, se comenzó a diseñar la aplicación. Se pensaron qué funciones proporcionaría la aplicación, cómo se presentarían estas al usuario y qué aspectos tecnológicos serían necesarios para poder desarrollarlas.

Una vez teniendo clara la estructura básica de qué y cómo se tenían que hacer las cosas, se procedió a la implementación. Los detalles de esta parte se encuentran expuestos detalladamente en el capítulo 4. Finalizada esta etapa, se realizaron pruebas generales respecto a la navegación a través de la aplicación teniendo en cuenta los conceptos de interacción, respuesta y transparencia. Estas pruebas permitieron conocer algunos aspectos que podían ser mejorados sin representar mucho trabajo adicional, como cambios en la GUI y algunas funciones que mejorarían la experiencia del usuario. Finalmente, se procedió a escribir toda la documentación necesaria, incluyendo esta memoria.

El coste temporal para el desarrollo de este proyecto fue de 814 horas de trabajo. Estas se encontraron distribuidas en un lapso de siete meses, donde la cantidad de horas por jornada de trabajo variaron según la disponibilidad de tiempo y otros factores personales ajenos a esta memoria.

A.II Costes de desarrollo

En esta sección se presentará un estudio básico sobre el coste estimado para el desarrollo de una aplicación de este tipo. Se tomarán en cuenta costes de implementación relacionados con el personal humano, equipamiento, lugar de trabajo y licencias de software y distribución.

Primero se tiene el coste que representa la remuneración salarial de un Ingeniero en Telecomunicaciones trabajando en el ámbito de la programación como desarrollador de aplicaciones Junior. Este perfil profesional registra un pago promedio de €12,5 la hora. Para calcular la cantidad de horas empleadas por el programador no se usarán los datos de la tabla A.1, ya que el perfil descrito requiere que el programador posea cierta experiencia en el desarrollo de aplicaciones móviles lo cual decrementaría

considerablemente las horas empleadas para el desarrollo de este proyecto. Estos valores temporales se muestran en la tabla A.2.

Tareas	Tiempo Invertido (horas)
Fase creativa	**30**
Estudio de las posibles plataformas	**12**
Diseño de la aplicación	**20**
Implementación	**164**
Captura y registro de información de historias	8
Visualizador de la historia	4
Grabador de audio PCM	56
Cabecera WAVE	16
Captura y registro de segmentos de historia	8
Visualizador de segmentos de historia	8
Reproductor de archivos de audio	12
Geolocalizador y etiquetas de posición	4
Herramientas de captura de audio	8
Visualizador de nuevos segmentos de historia	16
Transformador de voz y subida	12
Acceso y registro de usuarios	4
Cámara fotográfica	8
Pruebas	**16**
Depuración y mejoras	**24**
Documentación	**8**
Total	**274**

Tabla A.2. Estimación temporal de implementación de un desarrollador Junior.

Se puede observar que algunos valores, como el estudio de la plataforma y la fase de tutorialización, no se encuentran representados en esta tabla ya que no son de relevancia para un desarrollador con la experiencia suficiente. A su vez, el coste temporal de la fase documentación es sólo una pequeña fracción ya que el programador no tendrá que elaborar más que un informe técnico. Al final se obtendría un coste temporal estimado de 274 horas, lo cual se traduciría a un coste salarial de €3425. Este y el resto de los datos de coste se pueden observar en la tabla A.3

Las herramientas de software utilizadas para el desarrollo y la documentación final de este proyecto no representaron ningún coste ya que todas son de software libre.

En el caso de los equipos utilizados, un ordenador personal y un dispositivo móvil Android, no se tomaron en cuenta sus precios de compra, sino su depreciación durante el tiempo de desarrollo del proyecto. Esto se debe a que dichos equipos pueden ser utilizados para proyectos futuros, siendo inexacto atribuir la totalidad de su precio a los costes de implementación de una sola aplicación. La tasa de depreciación anual recomendada para activos fijos de este tipo es del 30% anual. Debido a que la duración del proyecto habría sido de aproximadamente dos meses, la tasa utilizada para este simple estudio fue de 5%. Por ende, tomando en cuenta un ordenador personal de un coste de 1000 euros y un dispositivo móvil de 255 euros, sus costes de depreciación son de aproximadamente €50 y €13 respectivamente.

Concepto	Coste [€]
Programador Junior	3.425
Herramientas de desarrollo	0
Android SDK	0
Wireshark 1.4.1	0
Eclipse IDE 3.5	0
Equipo	63
Opdenador personal	50
Dispositivo móvil	13
Licencia de desarrollador	18
Espacio de trabajo	103
Mobiliario	5
Electricidad	8
Alquiler	90
Total	**3.609**

Tabla A.3. Coste estimado del proyecto

En relación a los costes relacionados con el lugar de trabajo, se tomaron en cuenta el coste de alquiler del lugar, la depreciación del mobiliario de oficina y el consumo energético. El coste del alquiler de oficinas en una zona como 22@ en Barcelona puede

ascender a los 15€/m^2. Tomando en cuenta un espacio personal de 3m^2 y una duración del proyecto de dos meses, se traduce a un total €90. Por otra parte, la depreciación del mobiliario y equipo de oficina corresponde a un 10% anual. Tomando en cuenta como mobiliario una silla, un escritorio y un archivero para un total de €275, el coste implicado en el proyecto correspondería a €5. Por último, el consumo eléctrico de un ordenador en funcionamiento y luz está estimado en 30kWh/mes lo cual se traduciría a unos €8 teniendo en cuenta un coste de €0,14 kwh.

Por último se tiene la licencia de desarrollador. Para poder publicar una aplicación en el *Android Market*, o Mercado de Aplicaciones Android, se debe paga un único importe de 25 dólares, lo cual se traduce aproximadamente a unos 18 euros.

El coste total estimado para el desarrollo de esta aplicación es de aproximadamente 3.609 euros.

Anexo B

Aplicaciones Móviles como Negocio

En este capítulo se abarcarán brevemente algunos temas relacionados con el mundo de las aplicaciones móviles más allá de sus aspectos técnicos, se explicarán algunos de sus modelos de negocio y la posible rentabilidad económica detrás de esta emergente tecnología.

B.1 Modelos de Negocio

En este apartado se verán concretamente algunas de las formas más utilizadas de *cómo hacer dinero* en el mercado de las aplicaciones móviles, sus modalidades más comunes, dejando abierto un debate en este interesante y cambiante tema.

Hay que tener en cuenta que muchos de los usuarios de aplicaciones móviles provienen de una generación acostumbrada a obtener contenido digital cuando quieran y de manera gratuita. El paradigma de *si está en Internet es gratis* ha acompañado a los usuarios por casi veinte años, donde se descargaron millones de canciones en Napster, o películas en BitTorrent y donde se crean redes sociales con Facebook. Este es el principal reto de las aplicaciones móviles, e Internet en general, como negocio rentable. No obstante, existen muchas formas de atraer a los usuarios a pagar por un servicio o contenido, o

simplemente encontrar retribuciones económicas de manera paralela a un servicio sin lucro aparente.
Volviendo al tema concreto de las aplicaciones móviles se explicarán brevemente cuatro de los principales modelos de negocio más utilizados [41]:

Mercado de Aplicaciones

La manera más intuitiva de vender un producto, sea digital o no, es en una tienda. Los mercados de aplicaciones móviles ofrecen una vía poderosa para dar a conocer y vender aplicaciones móviles nativas de una plataforma específica. Los mercados ofrecen a los desarrolladores, ya sean compañías o usuarios independientes, elegir entre si cobrar o no a los usuarios por descargar sus aplicaciones. En el caso afirmativo, estos mercados cobran a los desarrolladores una comisión de un 30% por cada aplicación descargada, excepto Blackberry que cobra una comisión del 20% [54]. Aunque esta cifra puede parecer abultada, es importante tener en cuenta la exposición y credibilidad que ofrecen a las aplicaciones.

Un aspecto importante a tener en cuenta es que aunque estos mercados son efectivos, tal y como se puede ver en el capítulo anterior el desarrollo de aplicaciones nativas toma tiempo y dinero que en ocasiones no será compensado en ventas.

Suscripción por pago

Se basa en cobrar una cuota, generalmente mensual, para disfrutar de una aplicación que se considera especial a pesar que su descarga sea gratuita, es decir, el usuario paga por el servicio más que por la aplicación. Este modelo por lo general sigue una filosofía Freemium/Premium, donde se ofrece parte del servicio o producto de manera gratuita y se permite al usuario potenciarlo por una cuota. Aunque puede parecer absurdo cobrar por un producto después de ofrecerlo gratis, si las limitaciones de la versión gratuita en relación a las ventajas de la versión Premium son suficientes, los usuarios estarán dispuestos a dar el paso. Por otra parte, la viralidad que la aplicación obtiene al ofrecer una versión gratuita la hace llegar a más usuarios, transformándose en posibles futuras suscripciones y por ende ganancias.

Publicidad

Una de las principales ventajas de los dispositivos móviles es que ofrecen a sus usuarios información contextual relativa a sus necesidades y localización inmediatas. Esto puede ser utilizado por anunciantes ya que pueden llegar a los usuarios en el momento y lugar donde están más expuestos a realizar una decisión de compra. Adicionalmente, las aplicaciones móviles pueden vender a los anunciantes todo tipo de información acerca de sus usuarios y permitiendo realizar precisos estudios de mercado para dirigir de manera más efectiva la publicidad a los sectores deseados. Aunque generalmente publicidad por si sola no ofrece ganancias suficientes, es un modelo de negocio que hay que tomar en cuenta. Adicionalmente, es común ver trabajando juntos a este modelo y al de suscripción por pago donde una de las ventajas que trae potenciar la aplicación es la liberación de los a veces tediosos comerciales.

Bienes Virtuales

Este modelo se basa en vender bienes virtuales o intangibles a través de la aplicación. Aunque esto pueda ser difícil de imaginar, representan la mayor parte de las ganancias de la industria de los juegos sociales. La dinámica de este tipo de juegos consiste generalmente en la creación un avatar que existe en un mundo virtual compartido con los demás usuarios del juego. Adicionalmente, el usuario puede personalizar su espacio virtual adquiriendo bienes. Existen muchos ejemplos de aplicaciones, móviles y no, que siguen este modelo de negocio. *HabboHotel*[38], un juego social popular a principios de siglo, permitía a sus usuarios comprar muebles, cuadros y artículos electrónicos virtuales para personalizar las casas donde vivían sus avatares. *FarmVille*[30], una aplicación de *Facebook* con más de 82 millones de jugadores, permite comprar créditos virtuales para cambiarlos por bienes dentro de sus granjas. Según estudios realizados por Reuters, la industria de los juegos sociales que obtuvo un billón de dólares en 2009, espera cuadruplicar sus ventas para el 2013 [63].

B.2 Rentabilidad del Negocio

El rápido y viral crecimiento del mercado de aplicaciones móviles ha llevado a empresarios y programadores plantearse entrar en este aparentemente rentable negocio.

Tan sólo basta con pasearse por Internet y leer noticias y reseñas de algunas aplicaciones móviles en el mercado. Aunque, como se ha mencionado anteriormente, la mayoría de estas aplicaciones son de carácter lúdico los números no dejan de impresionar.

Un ejemplo modelo de esto es el juego para iPhone *Angry Birds*[57] que ha vendido más 12 millones de copias a $0.99 cada una y contabilizado más de 30 millones de descargas gratuitas desde su lanzamiento en diciembre de 2009[16]. Otra aplicación con resultados también impresionantes es el juego *Plants vs. Zombies*, el cual a pesar de su relativamente alto precio de $2.99 por copia, contabilizó más de 300 mil ventas en los primeros nueve días de su lanzamiento[19]. Cifras como estas se ven cada vez de manera más frecuente en los mercados de aplicaciones móviles de las diferentes plataformas, aunque hay que tener en cuenta que son pocas las aplicaciones que llegan a los titulares de los periódicos. Estos casos pertenecen únicamente a primer modelo de negocio descrito en el apartado anterior, quedando otras maneras de rentabilizar el desarrollo de aplicaciones móviles.

Aunque actualmente la publicidad representa tan solo alrededor de un 1% de las ganancias de un negocio que generó $5.2 billones en 2010, promete convertirse en los próximos años en un lucrativo modelo dentro del contexto económico de las aplicaciones móviles. Esto va de la mano con el hecho que las aplicaciones gratuitas formen el 85% del mercado, haciendo a los desarrolladores dependientes de maneras alternativas para generar ingresos [34].

A pesar de lo lucrativo que el negocio de aplicaciones móviles pueda parecer hay que tener en cuenta que se trata de un mercado altamente competitivo y cambiante donde la ganancia promedio por aplicación disminuye cada año. La clave está no sólo en tener un buen producto que vender sino en encontrar nuevos modelos de negocio y adaptarlos al servicio que se pretende prestar.

Anexo C

Requerimientos del sistema e Instalación recomendada para el desarrollo en Android

Este anexo muestra los requerimientos de sistema y pasos a seguir para llevar a cabo la instalación recomendada del SDK de Android y el entorno de desarrollo Eclipse según la documentación oficial de Android y la configuración utilizada en el desarrollo de este proyecto.

Antes de comenzar a instalar el SDK es necesario estar seguros que el ordenador de desarrollo cumpla con los requerimientos del sistema y tenga las herramientas de software necesarias.

Sistemas Operativos soportados:

- Windows XP (32-bits), Vista (32 o 64-bits), o Windows 7 (32 o 64-bits).
- Mac OS X 10.5.8 o más reciente (sólo x86).
- Linux (probado en Ubuntu Linux y Lucid Lynx).
    - Se requiere la librería C de GNU (glibc) versión 2.7 o más reciente.
    - Para Ubuntu Linux se requiere la versión 8.04 o más reciente.

Entorno de desarrollo Ecplise:

- Eclipse versión 3.5 (Galileo) o más reciente.
- Módulo JDT para Eclipse.
- Paquete Eclipse IDE para desarrolladores Java. Disponible gratuitamente en la web oficial de Eclipse. http://www.eclipse.org/downloads/.
- JDK 5 o JDK 6.
- Módulo ADT (Android Development Tools).

Una vez el ordenador de desarrollo cumpla con los requerimientos básicos de sistema, se deberá instalar el paquete inicial del SDK de Android. Este se puede conseguir en la página oficial de Android para desarrolladores. http://developer.android.com/sdk/index.html. Al instalar este paquete el sistema corroborará si posee el JDK necesario e instalará las herramientas iniciales del SDK.

Se recomienda desarrollar en el entorno de desarrollo Eclipse por lo cual es necesario instalar el módulo ADT para Android. Para ello, se debe iniciar Eclipse y seleccionar en el menú superior la opción de **Help > Install New Software**. Una vez ahí se debe añadir el repositorio donde se encuentran los diferentes paquetes de Android. https.//dl-ssl.google.com/androide/eclipse/. Después de haber seguido los pasos del proceso de instalación se deberán agregar los paquetes que aparecen en el diálogo de Available Software y se deberán aceptar los términos de la licencia y reiniciar Eclipse.

Una vez se tenga en entorno de desarrollo listo, se deberán agregar las plataformas y el resto de componentes de la SDK de Android. Para ello, desde Eclipse se deberá seleccionar en el menú superior la opción **Window > Android SDK and AVD Manager**. Esto abrirá un diálogo donde se deberá seleccionar el apartado de **Available packages**, donde aparecerá un listado de la plataformas y demás componentes de la SDK. Se deberán instalar todas aquellas versiones de la plataforma Android sobre las cuales de desea desarrollar, incluyendo los módulos de Google de ser necesario.

Para más detalles e información complementaria se recomienda visitar el sitio web oficial para desarrolladores de Android. http://developer.android.com/.

Anexo D

Código Fuente

En este capítulo se presentará el código fuente de la aplicación en su totalidad. Primero se mostrarán las clases Java que componen la implementación de las funcionalidades de la aplicación y, posteriormente, se presentarán los documentos XML para la representación de las vistas de cada una de las pantallas.

Este capítulo puede ser de especial ayuda para todos aquellos que deseen desarrollar aplicaciones en Android, ya que aquí encontrarán una importante cantidad de ejemplos que describen varios de los fundamentos de la plataforma.

La compartición de código es una de las principales formas de colaboración tecnológica y ha sido una excelente práctica que ha permitido la construcción de sistemas y estándares a través de Internet. Este proyecto debe mucho a las diferentes comunidades de programadores Android y sólo es justo que este haga lo mismo.

```java
package mtg.tales;

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

import android.app.Dialog;
import android.app.ListActivity;
import android.app.ProgressDialog;
import android.content.Context;
import android.graphics.drawable.Drawable;
import android.media.MediaPlayer;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.BaseAdapter;
import android.widget.Button;
import android.widget.GridView;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemLongClickListener;

public class AddScreen extends ListActivity{

    AudioRecorder myRecorder;
    static ArrayList<AddItem> addeditems = new ArrayList<AddItem>();
    boolean recording = false;
    int Tag = 99000;
    MediaPlayer mp = new MediaPlayer();
    File tempdir = new File(Environment.getExternalStorageDirectory()+"/temp/");
    MyGPS mGPS = new MyGPS();

    @Override
    public void onCreate(Bundle icicle){

      super.onCreate(icicle);
      setContentView(R.layout.addparts);

      mGPS.startGPS(this);

      try{
          tempdir.mkdirs();//create temp directory

          TextView StoryTitle = (TextView) this.findViewById(R.id.StoryTitle);
          StoryTitle.setText(DataRetriever.getTitle());

          ImageButton Record = (ImageButton) findViewById(R.id.RecordButton);
          Record.setOnClickListener(new ImageButton.OnClickListener(){

              @Override
              public void onClick(View arg0) {

                  if (!recording){
                      ChooseCharDialog();
```

```
65
66                  }else if(recording){
67                      recording = !recording;
68                      recordWave(recording);
69                      //To update the list after a new item is added
70                      ListUpdater();
71                  }
72              }
73          });
74
75          ImageButton Transform = (ImageButton) findViewById(R.id.TransformButton);
76          Transform.setOnClickListener(new ImageButton.OnClickListener(){
77
78              @Override
79              public void onClick(View arg0) {
80                  new TransformFiles().execute();//non-UI Thread
81              }
82          });
83
84          ListView AddPart = (ListView) findViewById(android.R.id.list);
85          AddPart.setAdapter(new AddPartAdapter(this, R.layout.part, addeditems));
86          AddPart.setOnItemLongClickListener(new OnItemLongClickListener(){
87
88              public boolean onItemLongClick(AdapterView<?> parent, View view, int
    position, long id) {
89
90                  File file = new File(addeditems.get(position).file+"");
91                  file.delete();
92                  addeditems.remove(position);
93                  ListUpdater();//update list on part deletion
94                  return true;
95              }
96          });
97
98          ImageButton UploadParts = (ImageButton) findViewById(R.id.UploadParts);
99          UploadParts.setOnClickListener(new ImageButton.OnClickListener(){
100
101             PartUploader Uploader = new PartUploader();
102
103             @Override
104             public void onClick(View arg0) {
105
106                 try {
107                     Uploader.contentbuilder();
108                     Uploader.upload();
109
110                     Toast success = new Toast(AddScreen.this);
111                     success = Toast.makeText(AddScreen.this, PartUploader.status,
    5000);
112                     success.show();
113
114                     String status = "Created";
115
116                     if ((PartUploader.status).equalsIgnoreCase(status)==true){
117                         removeDir();
118                         addeditems.clear();
119                         ListUpdater();
120                     }
121
122                 } catch (Exception e) {
123                     e.printStackTrace();
124                 }
125             }
126         });
```

```
127
128       }catch (Exception e){
129           e.printStackTrace();
130       }
131     }
132
133     public void removeDir(){
134
135         File[] tempfiles = tempdir.listFiles();
136
137         int i;
138         for (i=0; i< tempfiles.length; i++){
139             tempfiles[i].delete();
140         }
141     }
142
143     public void ListUpdater(){
144         //To update the list after a new a change
145         AddPartAdapter myAdapter =
146             new AddPartAdapter(AddScreen.this, android.R.id.list, addeditems);
147         AddScreen.this.setListAdapter(myAdapter);
148         myAdapter.notifyDataSetChanged();
149     }
150
151     public void ChooseCharDialog(){
152
153         final Dialog mDialog = new Dialog(this);
154         mDialog.setContentView(R.layout.choosechardialog);
155         mDialog.setTitle("Please choose a character");
156
157         GridView ChooseCharDialog= (GridView)
    mDialog.findViewById(R.id.CharacterLayout);
158         ChooseCharDialog.setAdapter(new CharacterAdapter(this));
159         ChooseCharDialog.setOnItemClickListener(new OnItemClickListener(){
160
161             @Override
162             public void onItemClick(AdapterView<?> parent, View view, int position,
    long id) {
163
164                 addNewPart();
165                 int count = addeditems.size();
166                 addeditems.get(count-1).character =
    DataRetriever.getCharacterAvatars(position);
167                 addeditems.get(count-1).voice =
    DataRetriever.getCharacterVoices(position);
168                 addeditems.get(count-1).character_name =
    DataRetriever.getCharacterNames(position);
169
170                 mDialog.dismiss();
171                 recording = !recording;
172                 recordWave(recording);
173             }
174         });
175
176         Button CancelButton = (Button) mDialog.findViewById(R.id.CancelButton);
177         CancelButton.setOnClickListener(new OnClickListener() {
178
179             public void onClick(final View v) {
180                 mDialog.cancel();
181             }
182         });
183
184         mDialog.show();
185     }
```

```
186
187     //To play files in the New Item list
188     public void onListItemClick (ListView parent, View v, int position, long id){
189
190         String datasource = null;
191
192         if (Tag != position  && mp.isPlaying() == false){
193             try{
194                 mp = new MediaPlayer();
195                 mp.setDataSource(addeditems.get(position).file);
196                 mp.prepare();
197                 mp.getDuration();
198                 mp.start();
199                 Tag = position;
200
201             }catch( Exception e){
202                 Log.v("MyPlayer", e+"");
203             }
204
205         }else if ((isPlaybackCompleted() || isPaused()) && Tag==position){
206             mp.start();
207
208         }else if (mp.isPlaying() && Tag == position){
209             mp.pause();
210
211         }if (Tag != position  && mp.isPlaying() == true){
212             try{
213                 mp.pause();
214                 mp = new MediaPlayer();
215                 mp.setDataSource(datasource);
216                 mp.prepare();
217                 mp.getDuration();
218                 mp.start();
219                 Tag = position;
220
221             }catch( Exception e){
222                 Log.v("MyPlayer", e+"");
223                 Log.v("MyPlayer", DataRetriever.getPartKey(position));
224             }
225         }
226     }
227
228     public boolean isPaused(){
229        if (mp.isPlaying() == false && isPlaybackCompleted()==false){
230            return true;
231        }else{
232            return false;
233        }
234    }
235
236     public boolean isPlaybackCompleted(){
237        if (mp.getCurrentPosition() == mp.getDuration()&& mp.isPlaying()== false){
238            return true;
239        }else{
240            return false;
241        }
242    }
243
244     public void recordWave(boolean recording){
245
246         Thread th = new Thread();
247         try {
248             if (recording){
249                 myRecorder = new AudioRecorder(120);
```

```
250                 th = new Thread(myRecorder);
251                 th.start();
252                 myRecorder.setRecording(true);
253 
254             }else if(!recording){
255                 myRecorder.setRecording(false);
256                 th.join();
257                 myRecorder.writeRawDataToFile("/temp/RAW.raw");
258                 WaveField myWave = new WaveField();
259                 myWave.WaveFile();
260             }
261 
262         } catch(Exception e) {
263             e.printStackTrace();
264         }
265     }
266 
267     public void addNewPart(){
268         AddItem newitem = new AddItem();
269         addeditems.add(newitem);
270     }
271 
272     //for the character's grid view
273     public class CharacterAdapter extends BaseAdapter {
274 
275         private Context mContext;
276 
277         public CharacterAdapter(Context c) {
278             mContext = c;
279         }
280 
281         public View getView(int position, View convertView, ViewGroup parent) {
282 
283             View view;
284             if (convertView == null) {
285 
286                 LayoutInflater inflater =
    (LayoutInflater)mContext.getSystemService(
287                     Context.LAYOUT_INFLATER_SERVICE);
288                 view = inflater.inflate(R.layout.character, null);
289 
290                 ImageView imageView = (ImageView)
    view.findViewById(R.id.CharacterAvatar);
291 
    imageView.setImageDrawable(DataRetriever.getCharacterAvatars(position));
292 
293                 TextView textView = (TextView)
    view.findViewById(R.id.CharacterName);
294                 textView.setText(DataRetriever.getCharacterNames(position));
295 
296             } else {
297                 view = convertView;
298             }
299 
300             return view;
301         }
302 
303         public int getCount() {
304             return DataRetriever.numcharacters + 1;//+1 is for the narrator
305         }
306 
307         public boolean areAllItemsEnabled() {
308             return false;
309         }
```

```
310
311         public boolean isEnabled(int position) {
312             return false;
313         }
314
315         public Object getItem(int position) {
316             return null;
317         }
318
319         public long getItemId(int position) {
320             return 0;
321         }
322     }
323
324     //for the new parts list
325     public class AddPartAdapter extends ArrayAdapter<AddItem> {
326
327         public AddPartAdapter(Context context, int textViewResourceId,
    ArrayList<AddItem> items) {
328             super(context, textViewResourceId, items);
329             addeditems = items;
330         }
331
332         @Override
333         public View getView(int position, View convertView, ViewGroup parent) {
334
335            View v = convertView;
336
337            if (convertView == null) {
338
339                LayoutInflater inflater = (LayoutInflater) getSystemService(
340                        Context.LAYOUT_INFLATER_SERVICE);
341                v = inflater.inflate(R.layout.part, null);
342
343                ImageView CharacterAvatar = (ImageView)
    v.findViewById(R.id.CharacterAvatar);
344
    CharacterAvatar.setImageDrawable(AddScreen.addeditems.get(position).character);
345
346                //Display Participant Photo
347                ImageView ParticipantAvatar = (ImageView)
    v.findViewById(R.id.ParticipantAvatar);
348                ParticipantAvatar.setImageResource(R.drawable.gadget);
349            }
350         return v;
351         }
352     }
353
354     private class TransformFiles extends AsyncTask <Void, Void, String> {
355
356         int badtransforms = 0;
357         ProgressDialog progress = new ProgressDialog(AddScreen.this);
358
359         protected void onPreExecute(){
360             //UI Thread
361             progress = ProgressDialog.show(AddScreen.this, "",
362                     "Transforming. Please wait...", false, true);
363         }
364
365         protected String doInBackground(Void...voids ) {
366
367             String completed;
368             String successful = "true";
369             TransformTask Transform = new TransformTask();
```

```
370
371             //non-UI Thread
372             try{
373                 int i = 0;
374                 for (i = 0 ; i < addeditems.size() ; i++){
375                     if (addeditems.get(i).voice.equals("-")== false
376                             && addeditems.get(i).transformed == false){
377                     //Check for Narrator or for previously transformed parts, if so
    do not Transform
378                         Transform.FileCollectionResource(addeditems.get(i).file+"");
379                         Transform.TaskCollectionResource(addeditems.get(i).voice);
380
381                         //Checks if the transformation is completed
382                         do{
383                             Thread.sleep(500);
384                             completed = Transform.Retriever("completed");
385                         }while(completed.equals("false"));
386
387                         if (Transform.Retriever("successful").equals("true")){
388                             addeditems.get(i).transformed = true;
389                             addeditems.get(i).file = DataRetriever.server +
    DataRetriever.files_url + Transform.Retriever("output_id") +"/conversion/mp3";
390                             addeditems.get(i).key =
    Transform.Retriever("output_id");
391
392                         }else if
     (Transform.Retriever("successful").equals("false")){
393                             //If just one file of the transform batch is
    unsuccessful
394                             //the whole process will be unsuccessful
395                             badtransforms = badtransforms +1;
396                             successful = "false";
397                         }
398                     }
399
400                     if (addeditems.get(i).voice.equals("-")== true
401                             && addeditems.get(i).transformed == false){
402
403                         Transform.FileCollectionResource(addeditems.get(i).file+"");
404
405                         do{
406                             Thread.sleep(500);
407
408                         }while(Transform.filekey == null);
409                         addeditems.get(i).transformed = true;
410                         addeditems.get(i).key = Transform.filekey;
411                     }
412                 }
413             }catch(Exception e){
414                 e.printStackTrace();
415             }
416             return successful;
417          }
418
419         protected void onPostExecute(String successful){
420
421             //UI Thread
422             progress.dismiss();
423
424             if (successful.equals("true")){
425                 ListUpdater();
426                 Toast success = new Toast(AddScreen.this);
427                 success = Toast.makeText(AddScreen.this, "Transform SUCCESSFUL",
    5000);
```

```
428                 success.show();
429
430             }else if (successful.equals("false")){
431                 Toast error = new Toast(AddScreen.this);
432                 error = Toast.makeText(AddScreen.this, "Transform UNSUCCESSFUL: "
433                         + badtransforms + " file(s) failed", 5000);
434                 error.show();
435             }
436         }
437     }
438
439     public class AddItem{
440
441         String country, city, voice, file, character_name, key;
442         Drawable participant, character;
443         boolean transformed, uploaded;
444
445         public AddItem(){
446
447             this.character = null;
448             this.participant = null;
449             this.country = null;
450             this.city = "Spain";
451             this.file = "Campeones!!!";
452             this.voice = null;
453             this.transformed = false;
454             this.uploaded = false;
455             this.character_name = null;
456
457         }
458     }
459 }
```

```java
package mtg.tales;

import java.io.BufferedOutputStream;

public class AudioRecorder implements Runnable {

    public  String TAG = "MyActivity";

    private int sampleRate = 44100;
    private int channelConfiguration = AudioFormat.CHANNEL_CONFIGURATION_MONO;
    private int audioEncoding = AudioFormat.ENCODING_PCM_16BIT;
    private int maxSeconds = 120;
    private int samplesRead;
    private Object mutex;

    private volatile boolean isRecording;

    private short[] tempBuffer;

    public AudioRecorder(int maxSeconds) {
        this.maxSeconds = maxSeconds;
        this.mutex = new Object();
        this.setRecording(false);
    }

    private int initializeTempBuffer() {
        int maxSamples = maxSeconds * this.sampleRate * 1;
        tempBuffer = new short[ maxSamples ]; // 1 for mono
        return maxSamples;
    }

    public void run() {
        // Wait until recording
        synchronized(mutex) {
            while (!this.getRecording()) {
                try {
                    mutex.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }

        android.os.Process.setThreadPriority
            (android.os.Process.THREAD_PRIORITY_URGENT_AUDIO);

        // Allocate Recorder and Start Recording
        int bufferSize = 2*AudioRecord.getMinBufferSize(this.sampleRate,
                                                        this.channelConfiguration,
                                                        this.audioEncoding);

        AudioRecord recorder = new AudioRecord(MediaRecorder.AudioSource.MIC,
                                               this.sampleRate,
                                               this.channelConfiguration,
                                               this.audioEncoding,
                                               bufferSize);

        int maxSamples = initializeTempBuffer();

        recorder.startRecording();

        samplesRead = 0;
        int  samplesToRead = bufferSize / 2;
        int  readResult;
```

```
 77
 78         while (this.getRecording()){
 79             // stop recording if we have no more space
 80             if(samplesRead + samplesToRead > maxSamples) {
 81                 this.setRecording(false);
 82                 continue;
 83             }
 84
 85             readResult = recorder.read(tempBuffer, samplesRead, samplesToRead);
 86
 87             if (readResult == AudioRecord.ERROR_INVALID_OPERATION ||
 88                 readResult == AudioRecord.ERROR_BAD_VALUE) {
 89                     this.setRecording(false);
 90                     continue;
 91             }
 92             samplesRead += readResult;
 93         }
 94         recorder.stop();
 95     }
 96
 97     public void writeRawDataToFile(String fileName)
 98         throws FileNotFoundException, IOException {
 99
100         File file = new File(Environment.getExternalStorageDirectory().
101                 getAbsolutePath(), fileName);
102         if (file.exists()) file.delete();
103         file.createNewFile();
104
105         BufferedOutputStream bufferedStreamInstance = null;
106         bufferedStreamInstance = new BufferedOutputStream
107             (new FileOutputStream(file));
108         DataOutputStream dataOutputStreamInstance =
109             new DataOutputStream(bufferedStreamInstance);
110
111         for (int idxBuffer = 0; idxBuffer < samplesRead; idxBuffer++) {
112             dataOutputStreamInstance.writeShort(endianSwitch(tempBuffer[idxBuffer])
    );
113         }
114         bufferedStreamInstance.close();
115     }
116
117     public void setRecording(boolean isRecording) {
118         synchronized(mutex) {
119             this.isRecording = isRecording;
120             if (this.isRecording) {
121                 mutex.notify();
122             }
123         }
124     }
125
126     public boolean getRecording() {
127         synchronized(mutex) {
128             return isRecording;
129         }
130     }
131
132     public short endianSwitch(short x) {
133         return ByteBuffer.allocate(8)
134                 .order(ByteOrder.BIG_ENDIAN).putShort(x)
135                 .order(ByteOrder.LITTLE_ENDIAN).getShort(0);
136     }
137 }
```

Character.java

```java
package mtg.tales;

import android.graphics.drawable.Drawable;

public class Character {

    String name, voice;
    int id;
    Drawable avatar;

    public Character(){
        this.id = 0;
        this.name = null;
        this.avatar = null;
        this.voice = null;
    }

    public Character ( int id, String name, Drawable avatar, String voice){

        id = this.id;
        name = this.name;
        avatar = this.avatar;
        voice = this.voice;

    }
}
```

Creator.java

```java
package mtg.tales;

import android.graphics.drawable.Drawable;

public class Creator {

    String name;
    String country;
    String city;
    int id;
    Drawable avatar;

    public Creator(){

        this.name = null;
        this.city = null;
        this.country = null;
        this.id = 0;
        this.avatar = null;

    }

    public Creator(int id, String name, String city, String country, Drawable avatar){

        this.id = id;
        this.name = name;
        this.city = city;
        this.country = country;
        this.avatar = avatar;

    }
}
```

DataRetriever.java

```java
package mtg.tales;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;

import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.content.Context;
import android.graphics.drawable.Drawable;
import android.net.Uri;

public class DataRetriever {

    private static String media_url = "http://mtg105.upf.es/site_media/";
    static String server = "http://mtg105.upf.es/";
    static String story_url = "api/tales/stories/";
    static String files_url = "api/files/";

    static int numcharacters;
    static int numparts;
    static int numparticipants;

    static JSONObject storydata;
    static JSONArray partsdata;

    Uri photo;

    static ArrayList<Character> characters = new ArrayList<Character>();
    static ArrayList<Part> parts = new ArrayList<Part>();
    static ArrayList<Participant> participant = new ArrayList<Participant>();

    Context mContext;
```

```
42
43    public static void getStoryData(int story_id) throws
  IllegalStateException, ClientProtocolException, IOException,
  JSONException{
44
45        HttpClient httpclient = new DefaultHttpClient();
46        String URL = server + story_url + story_id;
47        HttpGet GET_story = new HttpGet(URL);
48        InputStream response = httpclient.execute
  (GET_story).getEntity().getContent();
49        String result = StreamToString(response);
50        storydata = new JSONArray(result).getJSONObject(0);
51
52        //Story Title
53        Story.title = storydata.getString("title");
54
55        //Story Language
56        Story.language = storydata.getString("language");
57
58        //Story Settings
59        Story.setting = storydata.getString("setting");
60
61        //Story ID
62        Story.id = storydata.getInt("id");
63
64        //Story Creator's ID
65        Story.creator.id = storydata.getJSONObject
  ("created_by").getInt("id");
66
67        //Story Creator's Name
68        Story.creator.name = storydata.getJSONObject
  ("created_by").getString("username");
69
70        //Story Creator's Photo
71        String creator_avatar_url = media_url +
  storydata.getJSONObject("created_by").getJSONObject
  ("profile").getString("avatar");
72        HttpClient creator_client = new DefaultHttpClient();
73        HttpGet GET_creator = new HttpGet(creator_avatar_url);
74        InputStream creator_response = creator_client.execute
  (GET_creator).getEntity().getContent();
75        Story.creator.avatar = Drawable.createFromStream
  (creator_response, "photo");
76
```

```
77          //Stroy Characters
78          numcharacters = storydata.getJSONArray
    ("characters").length();//+1 for the Narrator
79
80          //Add characters to ArrayList
81          int position = 0;
82
83          for (position = 0; position <
    DataRetriever.numcharacters ; position++){
84
85              Character character = new Character();
86              Story.characters.add(character);
87
88              //ID
89              Story.characters.get(position).id =
    storydata.getJSONArray("characters")
90                  .getJSONObject(position).getInt("id");
91
92              //Name
93              Story.characters.get(position).name =
    storydata.getJSONArray("characters")
94                  .getJSONObject(position).getString("full_name");
95
96              //Voice
97              Story.characters.get(position).voice =
    storydata.getJSONArray("characters")
98                  .getJSONObject(position).getString("voice_type");
99
100             //Avatar
101             String character_avatar_url = media_url +
    storydata.getJSONArray("characters")
102                 .getJSONObject(position).getString("avatar");
103             HttpClient character_client = new DefaultHttpClient();
104             HttpGet GET_character = new HttpGet
    (character_avatar_url);
105             InputStream character_response =
    character_client.execute(GET_character).getEntity().getContent();
106             Story.characters.get(position).avatar =
    Drawable.createFromStream(character_response, "photo");
107
108         }
109
110         //Add Narrator manually
111         Character narrator = new Character();
```

```
112         Story.characters.add(numcharacters, narrator);
113 
114         //Name
115         Story.characters.get(numcharacters).name = "Narrator";
116 
117         //Avatar
118         String narrator_Url = media_url + "shakespeare.png";
119         HttpClient narrator_client = new DefaultHttpClient();
120         HttpGet GET_narrator = new HttpGet(narrator_Url);
121         InputStream response_narrator = narrator_client.execute
    (GET_narrator).getEntity().getContent();
122         Story.characters.get(numcharacters).avatar =
    Drawable.createFromStream(response_narrator, "narrator");
123 
124         //Voice
125         Story.characters.get(numcharacters).voice = "-";
126 
127         //ID
128         Story.characters.get(numcharacters).id = 118;
129 
130         //Participant
131         numparticipants = storydata.getJSONArray
    ("participants").length();
132         int user = 0;
133 
134         for (user = 0 ; user < DataRetriever.numparticipants ;
    user++){
135 
136             Participant participant = new Participant();
137             Story.participants.add(participant);
138 
139             //Avatar
140             String participant_avatar_url = media_url +
    storydata.getJSONArray("participants")
141                 .getJSONObject(user).getString("avatar");
142             HttpClient participant_client = new DefaultHttpClient
    ();
143             HttpGet GET_participant = new HttpGet
    (participant_avatar_url);
144             InputStream participant_response =
    participant_client.execute(GET_participant).getEntity().getContent
    ();
145             Story.participants.get(user).avatar =
    Drawable.createFromStream(participant_response, "photo");
```

```
146
147             //Name
148             Story.participants.get(user).name =
    storydata.getJSONArray("participants")
149                 .getJSONObject(user).getString("username");
150
151             //ID
152             Story.participants.get(user).id =
    storydata.getJSONArray("participants")
153             .getJSONObject(user).getInt("id");
154         }
155     }
156
157     //Part
158     public static void getPartData(int story_id) throws
    IllegalStateException, ClientProtocolException, IOException,
    JSONException{
159
160         HttpClient httpclient = new DefaultHttpClient();
161         String URL = server + story_url + story_id + "/parts";
162         HttpGet GET = new HttpGet(URL);
163         InputStream response = httpclient.execute(GET).getEntity
    ().getContent();
164         String result = StreamToString(response);
165         partsdata = new JSONArray(result);
166
167         numparts = partsdata.length();
168
169         int part_number = 0;
170         for (part_number = 0 ; part_number < numparts ;
    part_number++){
171
172             Part part = new Part();
173             parts.add(part);
174
175             //Characters' ID
176             parts.get(part_number).character =
    partsdata.getJSONObject(part_number).getInt("character");
177
178             //Participants' ID
179             parts.get(part_number).creator =
    partsdata.getJSONObject(part_number).getInt("creator");
180
181             //Parts' Keys
```

```
182             parts.get(part_number).key = partsdata.getJSONObject
    (part_number).getString("key");
183
184             //City
185             parts.get(part_number).city = partsdata.getJSONObject
    (part_number).getString("city");
186
187             //Country
188             parts.get(part_number).country =
    partsdata.getJSONObject(part_number).getString("country");
189         }
190         //Mapping...
191
192         int story_character = 0;
193         int part_character = 0;
194
195         for (part_character = 0 ; part_character <
    DataRetriever.numparts; part_character++){
196             for (story_character = 0 ; story_character <
    DataRetriever.numcharacters + 1; story_character++){
197                 if (parts.get(part_character).character ==
    Story.characters.get(story_character).id){
198                     parts.get(part_character).character_photo =
    Story.characters.get(story_character).avatar;
199                     parts.get(part_character).charname =
    Story.characters.get(story_character).name;
200                 }
201             }
202         }
203
204         int story_participant = 0;
205         int part_participant = 0;
206
207         for (part_participant = 0 ; part_participant <
    DataRetriever.numparts ; part_participant++){
208             for (story_participant = 0 ; story_participant <
    DataRetriever.numparticipants ; story_participant++){
209                 if (parts.get(part_participant).creator ==
    Story.participants.get(story_participant).id){
210                     parts.get(part_participant).creator_photo =
    Story.participants.get(story_participant).avatar;
211                     parts.get(part_participant).creator_name =
    Story.participants.get(story_participant).name;
212                 }
```

```
213             }
214         }
215     }
216
217     public static String getTitle(){
218
219         return Story.title;
220
221     }
222
223     public static String getLanguage(){
224
225         return Story.language;
226
227     }
228
229     public static String getSetting(){
230
231         return Story.setting;
232
233     }
234
235     public int getStoryID(){
236
237         return Story.id;
238
239     }
240
241     public int getCreatorID(){
242
243         return Story.creator.id;
244
245     }
246
247     public static String getCreatorName(){
248
249         return Story.creator.name;
250
251     }
252
253     public static Drawable getCreatorAvatar(){
254
255         return Story.creator.avatar;
256
```

```
257     }
258
259     public static int getParticipantID(int participant){
260
261         return Story.participants.get(participant).id;
262
263     }
264
265     public String getParticipantName(int participant){
266
267         return Story.participants.get(participant).name;
268
269     }
270
271     public static Drawable getParticipantAvatar(int participant){
272
273         return Story.participants.get(participant).avatar;
274
275     }
276
277     public static int getCharacterIDs(int character){
278
279         return Story.characters.get(character).id;
280
281     }
282
283     public static String getCharacterNames(int character){
284
285         return Story.characters.get(character).name;
286
287     }
288
289     public static String getCharacterVoices(int character){
290
291         return Story.characters.get(character).voice;
292
293     }
294
295     public static Drawable getCharacterAvatars(int position){
296
297         return Story.characters.get(position).avatar;
298
299     }
300
```

```
301     public static int getPartCharacterID (int position){
302 
303         return parts.get(position).character;
304 
305     }
306 
307     public static String getPartKey (int position){
308 
309         return parts.get(position).key;
310 
311     }
312 
313     public static int getPartCreatorID (int position){
314 
315         return parts.get(position).creator;
316 
317     }
318 
319     public static Drawable getPartParticipantDrawable(int
    position){
320 
321         return parts.get(position).creator_photo;
322     }
323 
324     public static String getPartParticipantName(int participant){
325 
326         return parts.get(participant).creator_name;
327     }
328 
329     public static Drawable getPartCharacterDrawable(int position){
330 
331         return parts.get(position).character_photo;
332 
333     }
334 
335     public static String getPartCharacterName(int position){
336 
337         return parts.get(position).charname;
338 
339     }
340 
341     //Complementary Methods
342     public static String StreamToString(InputStream response) {
343 
```

```
344         BufferedReader reader = new BufferedReader(new
    InputStreamReader(response));
345         StringBuilder sbuilder = new StringBuilder();
346         String line = null;
347 
348         try{
349 
350             while((line = reader.readLine()) != null){
351 
352                 sbuilder.append(line + "\n");
353 
354            }
355 
356         }catch (IOException e) {
357 
358             e.printStackTrace();
359 
360         }finally{
361 
362             try{
363 
364                 response.close();
365 
366             }catch (IOException e){
367 
368                 e.printStackTrace();
369             }
370         }
371 
372         return sbuilder.toString();
373     }
374 }
```

Login.java

```
 1 package mtg.tales;
 2
 3 import java.io.File;
22
23 public class Login extends Activity {
24
25     public static String user_email, user_login;
26     protected EditText Login, Email;
27
28     /** Called when the activity is first created. */
29     @Override
30     public void onCreate(Bundle savedInstanceState) {
31         super.onCreate(savedInstanceState);
32         setContentView(R.layout.login);
33
34         Login = (EditText)this.findViewById(R.id.Login_edit);
35         Email = (EditText)this.findViewById(R.id.Email_edit);
36
37         ImageButton Submit = (ImageButton) findViewById
   (R.id.Sumbit);
38         Submit.setOnClickListener(new ImageButton.OnClickListener
   (){
39
40                 @Override
41                 public void onClick(View arg0) {
42
43                     getEditText();
44                     try {
45                         sendUserInfo();
46                     } catch (UnsupportedEncodingException e) {
47                         e.printStackTrace();
48                     }
49                     Intent intent = new Intent(Login.this,
   StoryScreen.class);
50                     startActivity(intent);
51                 }
52         });
53
54         ImageView Participant = (ImageView) this.findViewById
   (R.id.ParticipantPhoto);
55
56         if (TakePhoto.mBitmap != null){
57
58             Participant.setImageBitmap(TakePhoto.mBitmap);
```

```
59
60          }
61
62          Participant.setOnClickListener(new
  ImageButton.OnClickListener(){
63
64              @Override
65              public void onClick(View arg0) {
66
67                  Intent intent = new Intent(Login.this,
  TakePhoto.class);
68                  startActivity(intent);
69
70              }
71          });
72      }
73
74      public void getEditText(){
75
76          user_login = Login.getText().toString();
77          user_email = Email.getText().toString();
78          Log.v("LoginTest", user_login+" : "+user_email);
79
80      }
81
82      public void sendUserInfo() throws UnsupportedEncodingException
  {
83
84          AbstractHttpClient client = new DefaultHttpClient();
85          String URL = "http://mtg105.upf.es/api/tales/accounts";
86          File file = new File
  (Environment.getExternalStorageDirectory()+"/photo.jpeg");
87          FileBody photo = new FileBody(file);
88          StringBody username = new StringBody(user_login);
89          StringBody email = new StringBody(user_email);
90          HttpPost POST = new HttpPost(URL);
91          MultipartEntity reqEntity = new MultipartEntity();
92          reqEntity.addPart("username", username);
93          reqEntity.addPart("photo", photo);
94          reqEntity.addPart("email", email);
95          POST.setEntity(reqEntity);
96
97          try {
98
```

```
 99             client.execute(POST);
100
101         } catch (Exception e) {
102
103             e.printStackTrace();
104         }
105     }
106 }
```

AndroidManifest.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/
  android"
3     package="mtg.tales"
4     android:versionCode="1"
5     android:versionName="1.0">
6 
7     <application android:icon="@drawable/icon"
  android:label="Tales" android:debuggable="true">
8 
9         <activity android:name=".Login"
10                 android:label="Login">
11             <intent-filter>
12                 <action android:name="android.intent.action.MAIN" /
  >
13                 <category
  android:name="android.intent.category.LAUNCHER" />
14             </intent-filter>
15         </activity>
16 
17         <activity android:name=".TakePhoto"
18                 android:label="TakePhoto">
19         </activity>
20 
21         <activity android:name=".StoryScreen"
22                 android:label="StoryScreen">
23         </activity>
24 
25         <activity android:name=".PartScreen"
26                 android:label="PartScreen">
27         </activity>
28 
29          <activity android:name=".AddScreen"
30                  android:label="AddScreen">
31         </activity>
32 
33     </application>
34     <uses-sdk android:minSdkVersion="3" />
35 
36     <uses-permission android:name="android.permission.INTERNET"></
  uses-permission>
37     <uses-permission
  android:name="android.permission.RECORD_AUDIO"></uses-permission>
38     <uses-permission android:name="android.permission.CAMERA"/>
```

AndroidManifest.xml

```
39          <uses-feature android:name="android.hardware.camera" />
40          <uses-feature
  android:name="android.hardware.camera.autofocus" />
41
42  </manifest>
```

```java
package mtg.tales;

import java.util.List;
import java.util.Locale;

import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;

import com.google.android.maps.GeoPoint;

public class MyGPS{

    private LocationManager mylocationmanager = null;
    private LocationListener mylocationlistener = null;
    private String mylocationprovider = LocationManager.NETWORK_PROVIDER;
    public GeoPoint coordenates;
    List<Address> addresses;
    public static String city, country;
    Context mContext;

    public void startGPS(Context context){
        mylocationmanager = (LocationManager)
  context.getSystemService(Context.LOCATION_SERVICE);
        mylocationlistener = new MyLocationListener();
        mylocationmanager.requestLocationUpdates(mylocationprovider, 0, 1000,
                mylocationlistener); //updated if the user moves 1km.

    }

    private class MyLocationListener implements LocationListener {

        public void onLocationChanged(Location loc) {
            if (loc != null) {
                coordenates = new GeoPoint((int) (loc.getLatitude() * 1E6),
                        (int) (loc.getLongitude() * 1E6));
                Log.v("Location", coordenates+"");
            }else{
                loc = mylocationmanager.getLastKnownLocation(mylocationprovider);
                coordenates = new GeoPoint((int) (loc.getLatitude() * 1E6),
                        (int) (loc.getLongitude() * 1E6));
                Log.v("Location", coordenates+"");
            }
            Geocoder geoCoder = new Geocoder(
                    mContext, Locale.getDefault());

            try {

                addresses = geoCoder.getFromLocation(
                        coordenates.getLatitudeE6()  / 1E6,
                        coordenates.getLongitudeE6() / 1E6, 1);
                Address address = addresses.get(0);
                city = address.getLocality();
                country = address.getCountryCode();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```

```
64
65          @Override
66          public void onProviderDisabled(String provider) {
67          }
68          @Override
69          public void onProviderEnabled(String provider) {
70          }
71          @Override
72          public void onStatusChanged(String provider, int status, Bundle extras) {
73          }
74      }
75
76      public static String getCity(){
77          return city;
78      }
79
80      public static String getCountry(){
81          return country;
82      }
83  }
```

Part.java

```java
package mtg.tales;

import android.graphics.drawable.Drawable;

public class Part {

    String key, charname, creator_name;
    int creator, character, lon, lat;
    Drawable creator_photo, character_photo;

    public Part(){

        this.creator_name = null;
        this.character = 0;
        this.creator = 0;
        this.key = null;
        this.creator_photo = null;
        this.charname = null;
        this.character_photo = null;
        this.lon = 0;
        this.lat = 0;

    }

    public Part(int character, Drawable character_photo, String charname, int creator, Drawable creator_photo, String creator_name, String key, int lon, int lat){

        this.creator_name = creator_name;
        this.charname = charname;
        this.character = character;
        this.character_photo = character_photo;
        this.creator = creator;
        this.creator_photo = creator_photo;
        this.key = key;
        this.lon = lon;
        this.lat = lat;
    }
}
```

Participant.java

```java
package mtg.tales;

import android.graphics.drawable.Drawable;

public class Participant {

    String name;
    int id;
    Drawable avatar;

    public Participant(){

        this.id = 0;
        this.name = null;
        this.avatar = null;
    }

    public Participant ( int id, String name, Drawable avatar){

        id = this.id;
        name = this.name;
        avatar = this.avatar;
    }
}
```

```
 1 package mtg.tales;
 2
 3 import java.io.InputStream;
28
29 public class PartScreen extends ListActivity{
30
31     MediaPlayer mp = new MediaPlayer();
32     int current_part = -1;
33     boolean play = false;
34     String[] mPartURL;
35     static String lock;
36     Drawable playchar, playparticipant;
37     String playcharname, playparticipantname;
38
39     Dialog mDialog;
40
41     @Override
42     public void onCreate(Bundle icicle){
43
44       super.onCreate(icicle);
45       setContentView(R.layout.storyparts);
46
47       try{
48
49           DataRetriever.getPartData(StoryScreen.story);
50
51           TextView StoryTitle = (TextView) this.findViewById
   (R.id.StoryTitle);
52           StoryTitle.setText(DataRetriever.getTitle());
53
54           ListView PartsList = (ListView) findViewById
   (android.R.id.list);
55           PartsList.setAdapter(new PartListAdapter(this));
56
57           ImageButton PlayAllButton = (ImageButton)
   this.findViewById(R.id.PlayButton);
58           PlayAllButton.setOnClickListener(new
   ImageButton.OnClickListener(){
59
60               @Override
61               public void onClick(View arg0){
62                   new PlayAllParts().execute();
63               }
64           });
```

```
65
66          ImageButton AddButton = (ImageButton) findViewById
    (R.id.AddButton);
67          AddButton.setOnClickListener(new View.OnClickListener()
    {
68
69                  @Override
70                  public void onClick(View v) {
71                      try{
72                          HttpClient httpclient = new
    DefaultHttpClient();
73                          String URL = DataRetriever.server +
    DataRetriever.story_url + StoryScreen.story + "/lock";
74                          HttpPost postlock = new HttpPost(URL);
75                          InputStream response = httpclient.execute
    (postlock).getEntity().getContent();
76                          lock = DataRetriever.StreamToString
    (response);
77
78                      }catch(Exception e){
79                          e.printStackTrace();
80                      }
81
82                      Intent intent = new Intent(PartScreen.this,
    AddScreen.class);
83                      startActivity(intent);
84                  }
85          });
86      }catch (Exception e){
87          e.printStackTrace();
88      }
89    }
90
91    public void onListItemClick (ListView parent, View v, int
    position, long id){
92
93        if (current_part != position  && mp.isPlaying() == false){
94            try{
95                mp.release();
96                mp = new MediaPlayer();
97                mp.setDataSource(DataRetriever.server +
    DataRetriever.files_url + DataRetriever.getPartKey(position) +"/
    conversion/mp3");
98                mp.prepare();
```

```
 99                mp.getDuration();
100                mp.start();
101                current_part = position;
102            }catch( Exception e){
103                e.printStackTrace();
104            }
105
106        }else if ((isPlaybackCompleted() || isPaused()) &&
   current_part==position){
107            mp.start();
108
109        }else if (mp.isPlaying() && current_part == position){
110            mp.pause();
111
112        }if (current_part != position  && mp.isPlaying() == true){
113            try{
114                mp.pause();
115                mp.release();
116                mp = new MediaPlayer();
117                mp.setDataSource(DataRetriever.server +
   DataRetriever.files_url + DataRetriever.getPartKey(position) +"/
   conversion/mp3");
118                mp.prepare();
119                mp.getDuration();
120                mp.start();
121                current_part = position;
122            }catch( Exception e){
123                e.printStackTrace();
124            }
125        }
126    }
127
128    public boolean isPaused(){
129
130       if (mp.isPlaying() == false && isPlaybackCompleted()
   ==false){
131            return true;
132        }else{
133            return false;
134        }
135   }
136
137    public boolean isPlaybackCompleted(){
138
```

```
139         if (mp.getCurrentPosition() == mp.getDuration()&&
    mp.isPlaying()== false){
140             return true;
141         }else{
142             return false;
143         }
144     }
145
146     public void dialogInflator(){
147
148         ImageView PartChar = (ImageView) mDialog.findViewById
    (R.id.PartChar);
149         PartChar.setImageDrawable(playchar);
150
151         TextView PartCharName = (TextView) mDialog.findViewById
    (R.id.PartCharName);
152         PartCharName.setText(playcharname);
153
154         ImageView PartParticipant = (ImageView)
    mDialog.findViewById(R.id.PartParticipant);
155         PartParticipant.setImageDrawable(playparticipant);
156
157         TextView PartParticipantName = (TextView)
    mDialog.findViewById(R.id.PartParticipantName);
158         PartParticipantName.setText(playparticipantname);
159     }
160
161     public void storytellingdialog(){
162
163         mDialog = new Dialog(this);
164         mDialog.setContentView(R.layout.storytellingdialog);
165         mDialog.setTitle("Once upon a time...");
166
167         dialogInflator();
168         Button CancelButton = (Button) mDialog.findViewById
    (R.id.CancelButton);
169         CancelButton.setOnClickListener(new OnClickListener() {
170
171             public void onClick(final View v) {
172                 mDialog.cancel();
173                 play = false;
174             }
175         });
176         mDialog.show();
```

```
177     }
178
179     //AsyncTask to run it in another Thread (in the background)
180     private class PlayAllParts extends AsyncTask <Drawable,
    Drawable, Object> {
181
182         protected void onPreExecute(){
183             play = true;
184             storytellingdialog();
185         }
186
187         protected Object doInBackground(Drawable...drawables ) {
188             try{
189                 int i;
190
191                 for (i = 0; i < DataRetriever.numparts; i++){
192                     if ( play == true){
193
194                         playchar =
    DataRetriever.getPartCharacterDrawable(i);
195                         playparticipant =
    DataRetriever.getPartParticipantDrawable(i);
196                         playcharname =
    DataRetriever.getPartCharacterName(i);
197                         playparticipantname =
    DataRetriever.getPartParticipantName(i);
198
199                         publishProgress(playchar,
    playparticipant);
200
201                         mp.release();
202                         mp = new MediaPlayer();
203                         mp.setDataSource(DataRetriever.server +
    DataRetriever.files_url + DataRetriever.getPartKey(i) +"/
    conversion/mp3");
204                         mp.prepare();
205                         mp.getDuration();
206                         mp.start();
207
208                         while (mp.isPlaying()){
209                             Thread.sleep(300);
210                         }
211                     }
212                 }
```

```
213
214             }catch (Exception e){
215                 e.printStackTrace();
216             }
217             return null;
218         }
219
220         protected void onProgressUpdate(Drawable... progress) {
221             dialogInflator();
222         }
223     }
224
225     public class PartListAdapter extends BaseAdapter {
226
227        private Context mContext;
228
229        public PartListAdapter (Context c){
230             mContext = c;
231         }
232
233        @Override
234        public View getView( int position, View convertView,
   ViewGroup parent) {
235
236            View view;
237
238            LayoutInflater inflater = (LayoutInflater)
   mContext.getSystemService(
239            Context.LAYOUT_INFLATER_SERVICE);
240            view = inflater.inflate(R.layout.part, null);
241
242            //Display Character Photo
243            ImageView CharacterAvatar = (ImageView)
   view.findViewById(R.id.CharacterAvatar);
244            CharacterAvatar.setImageDrawable
   (DataRetriever.getPartCharacterDrawable(position));
245
246            //Display Participant Photo
247            ImageView ParticipantAvatar = (ImageView)
   view.findViewById(R.id.ParticipantAvatar);
248            ParticipantAvatar.setImageDrawable
   (DataRetriever.getPartParticipantDrawable(position));
249
250            //Display Part City
```

```
251             TextView City = (TextView) view.findViewById
    (R.id.City);
252             City.setText(DataRetriever.getPartCity(position));
253 
254             //Display Part City
255             TextView Country = (TextView) view.findViewById
    (R.id.Country);
256             Country.setText(DataRetriever.getPartCity(position));
257 
258             return view;
259         }
260 
261         public boolean areAllItemsEnabled() {
262             return true;
263         }
264 
265         public boolean isEnabled(int position) {
266             return true;
267         }
268 
269         public int getCount() {
270             return DataRetriever.numparts;
271         }
272 
273         public Object getItem(int position) {
274             return null;
275         }
276 
277         public long getItemId(int position) {
278             return 0;
279         }
280 
281         @Override
282         public int getItemViewType(int arg0) {
283             return 0;
284         }
285 
286         @Override
287         public int getViewTypeCount() {
288             return 1;
289         }
290 
291         @Override
292         public boolean hasStableIds() {
```

```
293             return false;
294         }
295 
296         @Override
297         public boolean isEmpty() {
298             return false;
299         }
300 
301         @Override
302         public void registerDataSetObserver(DataSetObserver
    observer) {
303         }
304 
305         @Override
306         public void unregisterDataSetObserver(DataSetObserver
    observer) {
307         }
308     }
309 }
```

PartUploader.java

```java
package mtg.tales;

import java.io.IOException;

public class PartUploader {

    AbstractHttpClient client = new DefaultHttpClient();
    String user;
    String lock;
    String parts;
    static String status;

    public void upload() throws ClientProtocolException, IOException{

        String URL = DataRetriever.server + DataRetriever.story_url + StoryScreen.story + "/parts";
        StringBody user = new StringBody(this.user);
        StringBody lock = new StringBody(this.lock);
        StringBody parts = new StringBody(this.parts);
        HttpPost POST = new HttpPost(URL);
        MultipartEntity reqEntity = new MultipartEntity();
        reqEntity.addPart("user", user);
        reqEntity.addPart("parts", parts);
        reqEntity.addPart("lock", lock);
        POST.setEntity(reqEntity);
        InputStream response = client.execute(POST).getEntity().getContent();
        status = DataRetriever.StreamToString(response);
    }

    public void contentbuilder() throws JSONException{

        JSONStringer parts;
        user = Login.user_login;
        lock = PartScreen.lock;

        parts = new JSONStringer().array();

        int i = 0;
        for (i = 0 ; i < AddScreen.addeditems.size(); i++){

        parts.object()
                    .key("key").value(AddScreen.addeditems.get(i).key)

                .key("character").value(AddScreen.addeditems.get(i).character_name)
                        .key("country").value(MyGPS.getCountry())
                        .key("city").value(MyGPS.getCity())
                 .endObject();
        }
        parts.endArray();
        this.parts = parts.toString();
    }
}
```

Story.java

```java
package mtg.tales;

import java.util.ArrayList;

public class Story {

    static String language, title, setting;
    static Creator creator = new Creator();
    static ArrayList<Participant> participants = new ArrayList<Participant>();
    static ArrayList<Character> characters = new ArrayList<Character>();
    static int id;

    public Story(){

        language = null;
        title = null;
        setting = null;
        creator = null;
        participants = null;
        characters = null;
        id = 0;

    }

    public Story(String language, String title, String setting , Creator creator, ArrayList<Participant> participants, ArrayList<Character> characters, int id){

        Story.language = language;
        Story.title = title;
        Story.setting = setting;
        Story.creator = creator;
        Story.participants = participants;
        Story.characters = characters;
        Story.id = id;

    }
}

```

StoryScreen.java

```java
package mtg.tales;

import android.app.Activity;

public class StoryScreen extends Activity {

    final static int story = 124;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.story);

        try {

            DataRetriever.getStoryData(story);

            TextView StoryTitle = (TextView) this.findViewById(R.id.StoryTitle);
            StoryTitle.setText(DataRetriever.getTitle());

            TextView CreatorName = (TextView) this.findViewById(R.id.CreatorName);
            CreatorName.setText(DataRetriever.getCreatorName());

            TextView Settings = (TextView) this.findViewById(R.id.SettingsInfo);
            Settings.setText("'"+DataRetriever.getSetting()+"'");

            ImageView CreatorAvatar = (ImageView) this.findViewById(R.id.CreatorPhoto);
            CreatorAvatar.setImageDrawable(DataRetriever.getCreatorAvatar());

            GridView Characters = (GridView) findViewById(R.id.CharacterLayout);
            Characters.setAdapter(new CharacterAdapter(this));

            ImageButton GoToStory = (ImageButton) findViewById(R.id.GoToStory);
            GoToStory.setOnClickListener(new View.OnClickListener() {

                @Override
```

```
48                public void onClick(View v) {
49
50                    Intent intent = new Intent(StoryScreen.this,
  PartScreen.class);
51                    startActivity(intent);
52
53                }
54            });
55
56        } catch (Exception e) {
57            e.printStackTrace();
58        }
59    }
60
61    public class CharacterAdapter extends BaseAdapter {
62
63        private Context mContext;
64
65        public CharacterAdapter(Context c) {
66            mContext = c;
67        }
68
69        public View getView(int position, View convertView,
  ViewGroup parent) {
70
71            View view;
72
73            if (convertView == null) {
74
75                LayoutInflater inflater = (LayoutInflater)
  mContext.getSystemService(
76                    Context.LAYOUT_INFLATER_SERVICE);
77                view = inflater.inflate(R.layout.character, null);
78
79                ImageView imageView = (ImageView)
  view.findViewById(R.id.CharacterAvatar);
80                imageView.setImageDrawable
  (DataRetriever.getCharacterAvatars(position));
81
82                TextView textView = (TextView) view.findViewById
  (R.id.CharacterName);
83                textView.setText(DataRetriever.getCharacterNames
  (position));
84
```

```
85              } else {
86
87                  view = convertView;
88              }
89
90              return view;
91
92          }
93
94          public int getCount() {
95
96              return DataRetriever.numcharacters;
97          }
98
99          public boolean areAllItemsEnabled() {
100             return false;
101         }
102
103         public boolean isEnabled(int position) {
104             return false;
105         }
106
107         public Object getItem(int position) {
108             return null;
109         }
110
111         public long getItemId(int position) {
112             return 0;
113         }
114
115     }
116 }
```

```java
package mtg.tales;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.Bitmap.CompressFormat;
import android.graphics.BitmapFactory;
import android.graphics.PixelFormat;
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.os.Bundle;
import android.os.Environment;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.ImageButton;

public class TakePhoto extends Activity implements SurfaceHolder.Callback{

    Camera mCamera;
    static Bitmap mBitmap;

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

        getWindow().setFormat(PixelFormat.TRANSLUCENT);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.cameralayout);

        SurfaceView CameraView = (SurfaceView) findViewById(R.id.CameraView);
        CameraView.setOnClickListener(new
```

```
ImageButton.OnClickListener(){
42
43              @Override
44              public void onClick(View arg0) {
45
46                  mCamera.takePicture(mShutterCallback, null,
mJpegCallback);
47                  BitmapToSdcard();
48
49              }
50          });
51
52          ImageButton PhotoOK = (ImageButton) this.findViewById
(R.id.PhotoOK);
53          PhotoOK.setOnClickListener(new ImageButton.OnClickListener
(){
54
55              @Override
56              public void onClick(View arg0) {
57
58                  Intent intent = new Intent(TakePhoto.this,
Login.class);
59                  startActivity(intent);
60
61              }
62        });
63
64          SurfaceHolder mSurfaceHolder = CameraView.getHolder();
65          mSurfaceHolder.addCallback(this);
66          mSurfaceHolder.setSizeFromLayout();
67          mSurfaceHolder.setType
(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
68      }
69
70      @Override
71      public void surfaceChanged(SurfaceHolder holder, int format,
int width, int height) {
72
73          Camera.Parameters params = mCamera.getParameters();
74          params.setPreviewSize(width, height);
75          mCamera.setParameters(params);
76
77          try {
78
```

```
79              mCamera.setPreviewDisplay(holder);
80
81          } catch (IOException e) {
82              e.printStackTrace();
83          }
84
85          mCamera.startPreview();
86
87      }
88
89      @Override
90      public void surfaceCreated(SurfaceHolder holder) {
91
92          mCamera = Camera.open();
93      }
94
95      @Override
96      public void surfaceDestroyed(SurfaceHolder holder) {
97
98          mCamera.stopPreview();
99          mCamera.release();
100
101     }
102
103     Camera.PictureCallback mPictureCallback = new
    Camera.PictureCallback() {
104
105         public void onPictureTaken(byte[] imageData, Camera cam) {
106         }
107     };
108
109     ShutterCallback mShutterCallback = new ShutterCallback() {
110
111         public void onShutter() {
112
113         }
114     };
115
116     PictureCallback mJpegCallback = new PictureCallback() {
117
118         @Override
119         public void onPictureTaken(byte[] data, Camera camera) {
120
121             try {
```

```
122
123                 BitmapFactory.Options options = new
    BitmapFactory.Options();
124                 options.inSampleSize = 4;
125                 mBitmap = BitmapFactory.decodeByteArray(data, 0,
    data.length, options);
126                 mBitmap = mBitmap.copy(Bitmap.Config.RGB_565,
    false);
127
128             }catch (Exception e){
129
130                 e.printStackTrace();
131             }
132         }
133     };
134
135     public void BitmapToSdcard(){
136
137         try {
138             File file = new File
    (Environment.getExternalStorageDirectory()+"/photo.jpeg");
139             FileOutputStream outstream = new FileOutputStream
    (file);
140             mBitmap.compress(CompressFormat.JPEG, 55, outstream);
141             outstream.flush();
142             outstream.close();
143
144         }catch(Exception e){
145             e.printStackTrace();
146         }
147     }
148 }
```

```java
package mtg.tales;

import java.io.File;

public class TransformTask{

    AbstractHttpClient client = new DefaultHttpClient();
    String filekey, taskkey;
    InputStream jsonresponse;

    public void FileCollectionResource(String filepath)throws JSONException, IllegalStateException, ClientProtocolException, IOException{

        File file = new File(filepath);
        String URL = "http://mtg105.upf.es/api/files/";
        HttpPost POST = new HttpPost(URL);

        FileBody bin = new FileBody(file);
        StringBody license = new StringBody("cc_at_nc");

        MultipartEntity reqEntity = new MultipartEntity();
        reqEntity.addPart("license", license);
        reqEntity.addPart("file", bin);
        POST.setEntity(reqEntity);

        this.jsonresponse = client.execute(POST).getEntity().getContent();
        String response = DataRetriever.StreamToString(jsonresponse);
        this.filekey = new JSONObject(response).getString("_id");

        Log.v("MyActivity 1", this.filekey+"");

    }

    public void TaskCollectionResource(String transform) throws IllegalStateException, ClientProtocolException, IOException, JSONException{

        String URL = "http://mtg105.upf.es/api/processing/voicetransform/";
        StringBody transformation = new StringBody(transform);
        StringBody key = new StringBody(this.filekey);
        HttpPost POST = new HttpPost(URL);

        MultipartEntity reqEntity = new MultipartEntity();
        reqEntity.addPart("preset", transformation);
        reqEntity.addPart("input", key);
        POST.setEntity(reqEntity);

        InputStream jsonresponse = client.execute(POST).getEntity().getContent();
        String response = "{key: " + DataRetriever.StreamToString(jsonresponse) + "}";
        this.taskkey = new JSONObject(response).getString("key");

        Log.v("MyActivity 2", this.taskkey+"");

    }

    public String Retriever (String item) throws IllegalStateException, ClientProtocolException, IOException, JSONException{

        HttpClient httpclient = new DefaultHttpClient();
        String URL = "http://mtg105.upf.es/api/processing/voicetransform/" + this.taskkey;
        HttpGet GETstory = new HttpGet(URL);
        InputStream response = httpclient.execute(GETstory).getEntity().getContent();
```

```
75          String result = DataRetriever.StreamToString(response);
76          String value = new JSONObject(result).getString(item);
77
78          return value;
79
80      }
81
82  }
```

User.java

```java
package mtg.tales;

import android.graphics.drawable.Drawable;

public class User {

    static String name;
    static int id;
    static Drawable avatar;

    public User(){

        User.id = 0;
        User.name = null;
        User.avatar = null;
    }

    public User( int id, String name, Drawable avatar){

        User.id = id;
        User.name = name;
        User.avatar = avatar;
    }
}

```

```java
package mtg.tales;

import java.io.File;

public class WaveField{

    AudioRecorder myRecorder;
    static List<File> files = new ArrayList<File>();
    File rawfile = new File(Environment.getExternalStorageDirectory()
  +"/temp/RAW.raw");
    String path;

    int headerLength = 44;
    int mAudioFormat = 1;//PCM: Linear quantization
    int mSampleRate = 8000;
    int mBitsPerSample = 16;//16 bits
    int mSubchunk1Size = 16;// for PCM
    int mNumChannels = 1;//Mono
    int mSubchunk2Size;
    int mChunkSize, mNumSamples, mByteRate, mBlockAlign;

    byte[] bChunkSize = new byte[4];
    byte[] bByteRate = new byte[4];
    byte[] bSubchunk2ID = new byte[4];
    byte[] bChunkID = new byte[4];
    byte[] bSubchunk2Size = new byte[4];
    byte[] bFormat = new byte[4];
    byte[] bSubchunk1ID = new byte[4];
    byte[] bSubchunk1Size = new byte[4];
    byte[] bSampleRate = new byte[4];
    byte[] bAudioFormat = new byte[2];
    byte[] bNumChannels = new byte[2];
    byte[] bBitsPerSample = new byte[2];
    byte[] bBlockAlign = new byte[2];
    byte[] header = new byte[headerLength];

    public WaveField() throws UnsupportedEncodingException{
        super();
        this.setChunkID("RIFF");
        this.setChunkSize(mSubchunk2Size);
        this.setFormat("WAVE");
        this.setSubchunk1ID("fmt ");
        this.setSubchunk1Size(16);
        this.setAudioFormat(mAudioFormat);
        this.setNumChannels(mNumChannels);
        this.setSampleRate(mSampleRate);
        this.setByteRate(mSampleRate, mNumChannels, mBitsPerSample);
        this.setBlockAlign(mNumChannels, mBitsPerSample);
        this.setBitsPerSample(mBitsPerSample);
        this.setSubchunk2ID("data");
        this.setSubchunk2Size();
    }

    //Chunk
    public void setChunkID(String ChunkID) throws UnsupportedEncodingException {
        bChunkID = ChunkID.getBytes("Ascii");
        headerBuilderBE(bChunkID, 4, 0);
    }

    public void setChunkSize(int Subchunk2Size) {
        bChunkSize = intToByteArray(36 + Subchunk2Size);
        headerBuilderLE(bChunkSize, 4, 4);
    }

```

```
 73     //Format
 74     public void setFormat(String Format) throws UnsupportedEncodingException{
 75         bFormat = Format.getBytes("Ascii");
 76         headerBuilderBE(bFormat, 4, 8);
 77     }
 78 
 79     //Subchunk1
 80     public void setSubchunk1ID(String Subchunk1ID) throws
   UnsupportedEncodingException {
 81         bSubchunk1ID = Subchunk1ID.getBytes("Ascii");
 82         headerBuilderBE(bSubchunk1ID, 4, 12);
 83     }
 84 
 85     public void setSubchunk1Size(int mSubchunk1Size) {
 86         bSubchunk1Size = intToByteArray(mSubchunk1Size);
 87         headerBuilderLE(bSubchunk1Size, 4, 16);
 88     }
 89 
 90     //AudioFormat
 91     public void setAudioFormat(int mAudioFormat) {
 92         bAudioFormat = intTo2Byte(mAudioFormat);
 93         headerBuilderLE(bAudioFormat, 2, 20);
 94     }
 95 
 96     //NumChannels
 97     public void setNumChannels(int mNumChannels) {
 98         bNumChannels = intTo2Byte(mNumChannels);
 99         headerBuilderLE(bNumChannels, 2, 22);
100     }
101 
102     //SampleRate
103     public void setSampleRate(int mSampleRate) {
104         bSampleRate = intToByteArray(mSampleRate);
105         headerBuilderLE(bSampleRate, 4, 24);
106     }
107 
108     //ByteRate
109     public void setByteRate(int SampleRate, int NumChannels, int BitsPerSample) {
110         bByteRate = intToByteArray(SampleRate * NumChannels * BitsPerSample/8);
111         headerBuilderLE(bByteRate, 4, 28);
112     }
113 
114     //BlockAlign
115     public void setBlockAlign(int NumChannels, int BitsPerSample) {
116         bBlockAlign = intTo2Byte(NumChannels + BitsPerSample/8);
117         headerBuilderLE(bBlockAlign, 2, 32);
118     }
119 
120     //BitsPerSample
121     public void setBitsPerSample(int mBitsPerSample){
122         bBitsPerSample = intTo2Byte(mBitsPerSample);
123         headerBuilderLE(bBitsPerSample, 2, 34);
124     }
125 
126     //Subchunk2
127     public void setSubchunk2ID(String Subchunk2ID) throws
   UnsupportedEncodingException {
128 
129         bSubchunk2ID = Subchunk2ID.getBytes("Ascii");
130         headerBuilderBE(bSubchunk2ID, 4, 36);
131     }
132 
133     public void setSubchunk2Size() {
134         mSubchunk2Size = (int)rawfile.length();
```

```
135         bSubchunk2Size = intToByteArray(mSubchunk2Size);
136         headerBuilderLE(bSubchunk2Size, 4, 40);
137     }
138 
139     //Header in Big-endian
140     public void headerBuilderBE (byte[] array, int length, int offset){
141         int i;
142         for (i = 0 ; i < length ; i++){
143             header[i+offset] = array[i];
144         }
145     }
146 
147     //Header in Little-endian
148     public void headerBuilderLE (byte[] array, int length, int offset){
149         int i;
150         for (i = 0 ; i < length ; i++){
151             header[offset+length-1-i] = array[i];
152         }
153     }
154 
155     public static final byte[] intToByteArray(int value) {
156         return new byte[] {
157             (byte)(value >>> 24),
158             (byte)(value >>> 16),
159             (byte)(value >>> 8),
160             (byte)value
161         };
162     }
163 
164     public static final byte[] intTo2Byte(int value) {
165         return new byte[] {
166             (byte)(value >>> 8),
167             (byte)value
168         };
169     }
170 
171     public void WaveFile() throws IOException{
172 
173         //adding header to the wave array
174         byte[] wavearray = new byte[headerLength + mSubchunk2Size];
175         int i = 0;
176         for (i = 0; i < headerLength ; i++){
177             wavearray[i] = header[i];
178         }
179 
180         //reading the .raw audio from sdcard and storing in wave array
181         FileInputStream Fin = new FileInputStream(rawfile);
182         Fin.read(wavearray, headerLength, mSubchunk2Size);
183 
184         //passing the wavearray to the wavefile in sdcard
185         int count = AddScreen.addeditems.size();
186         Random r = new Random();
187         int filename = r.nextInt(99000);
188         AddScreen.addeditems.get(count-1).file =
189             new File(Environment.getExternalStorageDirectory()+"/temp/"+ filename
    +".wav").toString();
190         FileOutputStream Fout =
191             new FileOutputStream(new File(Environment.getExternalStorageDirectory()
    +"/temp/"+ filename +".wav"));
192         Fout.write(wavearray);
193     }
194 }
```

addparts.xml

```xml
<?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="wrap_content"
    android:paddingLeft="10dip"
    android:paddingRight="10dip"
    android:paddingBottom="10dip"
    android:background="#FFFFFF">

    <TextView
        android:id="@+id/StoryTitle"
        android:text="StoryTitle"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="10dip"
        android:paddingLeft="10dip"
        android:paddingRight="10dip"
        android:background="@drawable/title_bgrd"
        android:gravity="center"
        android:textSize = "25sp"
        android:textColor = "#ffffff"/>

    <LinearLayout
        android:id="@+id/RecordButtons"
        android:layout_below="@id/StoryTitle"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <ImageButton
            android:id="@+id/RecordButton"
            android:src="@drawable/record"
            android:background="@drawable/buttonselector"
            android:padding="5dip"
            android:adjustViewBounds="true"
            android:scaleType="fitXY"
            android:layout_height="85dip"
            android:layout_width="wrap_content"/>

        <ImageButton
            android:id="@+id/TransformButton"
            android:src="@drawable/transform"
            android:background="@drawable/buttonselector"
```

```
45              android:adjustViewBounds="true"
46              android:scaleType="fitXY"
47              android:layout_height="70dip"
48              android:layout_width="wrap_content"/>
49
50      </LinearLayout>
51
52      <ListView
53          android:id="@android:id/list"
54          android:layout_below="@id/RecordButtons"
55          android:layout_width="fill_parent"
56          android:layout_height="160dip"
57          android:divider="#000000"
58          android:dividerHeight="1dip"
59          android:headerDividersEnabled="true"
60          android:footerDividersEnabled="true"
61          android:focusable="true"
62          android:fadingEdge="horizontal"/>
63
64      <ImageButton
65          android:id="@+id/UploadParts"
66          android:src="@drawable/addit"
67          android:background="@drawable/buttonselector"
68          android:layout_below="@android:id/list"
69          android:layout_alignRight="@android:id/list"
70          android:paddingRight="5dip"
71          android:scaleType="fitXY"
72          android:adjustViewBounds="true"
73          android:layout_height="150dip"
74          android:layout_width="wrap_content"/>
75
76  </RelativeLayout>
```

cameralayout.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="horizontal"
  android:background="#FFFFFF">

    <SurfaceView
        android:id="@+id/CameraView"
        android:layout_margin="10dip"
        android:layout_width="360dip"
        android:layout_height="fill_parent"/>

    <ImageButton
        android:id="@+id/PhotoOK"
        android:src="@drawable/ok_pic"
        android:background="@drawable/empty"
        android:layout_centerInParent="true"
        android:paddingTop="20dip"
        android:paddingRight="10dip"
        android:adjustViewBounds="true"

        android:scaleType="fitXY"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"/>

</LinearLayout>

```

character.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:id="@+id/CharacterTemplate"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:gravity="center_horizontal"
    android:background="#ffffff">

    <ImageView
        android:id="@+id/CharacterAvatar"
        android:src="@drawable/icon"
        android:padding="5dip"
        android:adjustViewBounds="true"
        android:scaleType="fitCenter"
        android:layout_height="60dip"
        android:layout_width="60dip"
        android:clickable="true"/>

    <TextView
        android:id="@+id/CharacterName"
        android:text="Character Name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:paddingBottom="5dip"
        android:textSize = "12sp"
        android:textColor = "#000000"/>

</LinearLayout>

```

choosechardialog.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/
android"
    android:id="@+id/Layout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dip"
    android:background="#ffffff">

<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">

    <GridView
        android:id="@+id/CharacterLayout"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:columnWidth="60dp"
        android:verticalSpacing="5dp"
        android:horizontalSpacing="5dp"
        android:numColumns="4"/>

    <Button
        android:id="@+id/CancelButton"
        android:text="Cancel"
        android:layout_below="@id/CharacterLayout"
        android:layout_alignRight="@id/CharacterLayout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</RelativeLayout>

</ScrollView>

```

listselector.xml

```
<selector xmlns:android="http://schemas.android.com/apk/res/
android">
  <item android:state_enabled="false" android:color="#333" />
  <item android:state_window_focused="false" android:color="#CCC" />
  <item android:state_pressed="true" android:color="#FFF" />
  <item android:state_selected="true" android:color="#FFF" />
  <item android:color="#FF3" /> <!-- not selected -->
</selector>
```

login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/
android"
    android:id="@+id/Layout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dip"
    android:background="#ffffff">

<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="vertical"
      android:background="#FFFFFF">

    <ImageView
        android:id="@+id/Header"
        android:layout_width="wrap_content"
        android:layout_height="90dip"
        android:src="@drawable/login_bgrd"
        android:scaleType="fitXY"
        android:gravity="center"/>

    <TextView
        android:id="@+id/Login"
        android:text="Login"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="20dip"
        android:textColor="#000000"/>

    <EditText
        android:id="@+id/Login_edit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="20dip"
        android:textColor="#000000"/>

    <TextView
        android:id="@+id/Email"
        android:text="e-mail"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```
44          android:textSize="20dip"
45          android:textColor="#000000"/>
46
47      <EditText
48          android:id="@+id/Email_edit"
49          android:layout_width="fill_parent"
50          android:layout_height="wrap_content"
51          android:textSize="20dip"
52          android:textColor="#000000"/>
53
54      <ImageButton
55          android:id="@+id/ParticipantPhoto"
56          android:src="@drawable/no_image"
57          android:layout_gravity="center"
58          android:layout_margin="15dip"
59          android:scaleType="fitXY"
60          android:layout_width="120dip"
61          android:layout_height="wrap_content"
62          android:adjustViewBounds="true"/>
63
64      <ImageButton
65          android:id="@+id/Sumbit"
66          android:src="@drawable/login"
67          android:layout_gravity="center"
68          android:background="@drawable/buttonselector"
69          android:paddingTop="5dip"
70          android:adjustViewBounds="true"
71          android:scaleType="fitXY"
72          android:layout_height="wrap_content"
73          android:layout_width="wrap_content"/>
74
75  </LinearLayout>
76  </ScrollView>
77
```

part.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:id="@+id/PartTemplate"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal"
    android:paddingLeft="15dip"
    android:gravity="center"
    android:background="#ffffff">

   <ImageView
        android:id="@+id/CharacterAvatar"
        android:src="@drawable/shakespeare"
        android:paddingLeft="5dip"
        android:paddingRight="5dip"
        android:adjustViewBounds="true"
        android:scaleType="fitCenter"
        android:layout_height="45dip"
        android:layout_width="45dip"
        android:focusable="false"/>

   <ImageView
        android:id="@+id/ParticipantAvatar"
        android:src="@drawable/icon"
        android:paddingLeft="5dip"
        android:paddingRight="5dip"
        android:adjustViewBounds="true"
        android:scaleType="fitCenter"
        android:layout_height="45dip"
        android:layout_width="45dip"
        android:focusable="false"/>

   <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical"
        android:gravity="center"
        android:background="#ffffff"
        android:focusable="false">
       <TextView
            android:id="@+id/City"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:paddingTop="5dip"
```

```
            android:paddingLeft="5dip"
            android:gravity="left"
            android:textSize = "15sp"
            android:textColor = "#000000"
            android:focusable="false"/>
        <TextView
            android:id="@+id/Country"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:paddingBottom="5dip"
            android:paddingLeft="5dip"
            android:gravity="left"
            android:textSize = "12sp"
            android:textColor = "#000000"
            android:focusable="false"/>
    </LinearLayout>
</LinearLayout>
```

story.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/Layout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dip"
    android:background="#ffffff">

    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/StoryLayout"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:padding="10dip"
        android:background="#ffffff">

    <!-- Story Title -->
        <TextView
            android:id="@+id/StoryTitle"
            android:text="StoryTitle"

            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:paddingLeft="10dip"
            android:paddingRight="10dip"
            android:paddingBottom="10dip"

            android:background="@drawable/title_bgrd"
            android:gravity="center"
            android:textSize = "25sp"
            android:textColor = "#ffffff"/>


    <!-- Creator -->
        <ScrollView android:layout_toLeftOf="@+id/StoryTitle" android:layout_height="wrap_content" android:id="@+id/ScrollView01" android:layout_width="wrap_content"></ScrollView><TextView
            android:id="@+id/CreatorTag"
            android:text="Created by:"
            android:layout_below="@+id/StoryTitle"

            android:layout_width="fill_parent"
```

```
40              android:layout_height="wrap_content"
41              android:paddingBottom="5dip"
42
43              android:textSize="14sp"
44              android:textColor = "#000000"/>
45
46          <ImageView
47              android:id="@+id/CreatorPhoto"
48              android:src="@drawable/icon"
49              android:layout_below="@+id/CreatorTag"
50
51              android:paddingRight="5dip"
52              android:paddingBottom="5dip"
53              android:layout_width="wrap_content"
54              android:layout_height="wrap_content"
55              android:adjustViewBounds="true"
56              android:scaleType="fitXY"
57              android:maxHeight="80dip"/>
58
59           <TextView
60              android:id="@+id/CreatorName"
61              android:text="Creator Name"
62              android:layout_below="@id/CreatorTag"
63              android:layout_toRightOf="@+id/CreatorPhoto"
64              android:layout_alignTop="@id/CreatorPhoto"
65
66              android:layout_width="fill_parent"
67              android:layout_height="wrap_content"
68
69              android:textSize = "20sp"
70              android:textColor = "#000000"/>
71
72          <TextView
73              android:id="@+id/CreatorLocation"
74              android:text="Location"
75              android:layout_below="@id/CreatorName"
76              android:layout_toRightOf="@+id/CreatorPhoto"
77
78              android:layout_width="fill_parent"
79              android:layout_height="wrap_content"
80              android:paddingBottom="5dip"
81
82              android:textSize = "14sp"
83              android:textColor = "#000000"/>
```

```
84
85      <!-- Settings -->
86          <TextView
87              android:id="@+id/SettingTag"
88              android:text="Settings:"
89              android:layout_below="@id/CreatorPhoto"
90
91              android:layout_width="fill_parent"
92              android:layout_height="wrap_content"
93              android:paddingBottom="5dip"
94
95              android:textSize = "14sp"
96              android:textColor = "#000000"/>
97
98          <TextView
99              android:id="@+id/SettingsInfo"
100             android:text="'Short description of the story plot'"
101             android:hint="No settings available for this story"
102
103             android:layout_below="@+id/SettingTag"
104             android:layout_width="fill_parent"
105             android:layout_height="wrap_content"
106
107             android:paddingBottom="10dip"
108             android:gravity="center"
109             android:textSize = "12sp"
110             android:textColor = "#000000"/>
111
112     <!-- Characters -->
113         <TextView
114             android:id="@+id/CharactersTag"
115             android:text="Characters:"
116             android:layout_below="@id/SettingsInfo"
117
118             android:layout_width="fill_parent"
119             android:layout_height="wrap_content"
120             android:paddingBottom="5dip"
121
122             android:textSize = "14sp"
123             android:textColor = "#000000"/>
124
125         <GridView
126
127             android:id="@+id/CharacterLayout"
```

```
128             android:layout_below="@id/CharactersTag"
129
130             android:layout_width="fill_parent"
131             android:layout_height="wrap_content"
132             android:columnWidth="60dp"
133             android:verticalSpacing="5dp"
134             android:horizontalSpacing="5dp"
135             android:numColumns="4"/>
136
137         <ImageButton
138
139             android:id="@+id/GoToStory"
140             android:src="@drawable/gotostory_button"
141             android:layout_below="@id/CharacterLayout"
142             android:layout_alignParentRight="true"
143
144             android:background="@drawable/buttonselector"
145             android:adjustViewBounds="true"
146             android:scaleType="fitXY"
147             android:layout_height="100dip"
148             android:layout_width="wrap_content"
149             android:focusable="true"
150             android:focusableInTouchMode="true"
151             android:clickable="true"/>
152
153     </RelativeLayout>
154
155 </ScrollView>
```

storyparts.xml

```xml
<?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="wrap_content"
    android:paddingLeft="10dip"
    android:paddingRight="10dip"
    android:paddingBottom="10dip"
    android:background="#FFFFFF">

    <TextView
        android:id="@+id/StoryTitle"
        android:text="StoryTitle"

        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="10dip"
        android:paddingLeft="10dip"
        android:paddingRight="10dip"

        android:background="@drawable/title_bgrd"
        android:gravity="center"
        android:textSize = "25sp"
        android:textColor = "#ffffff"/>

    <ListView
        android:id="@android:id/list"
        android:layout_below="@id/StoryTitle"
        android:layout_width="fill_parent"
        android:layout_height="250dip"
        android:divider="#000000"
        android:dividerHeight="1dip"
        android:headerDividersEnabled="true"
        android:footerDividersEnabled="true"
        android:focusable="true"
        android:clickable="true"
        android:fadingEdge="horizontal"
        android:background="@drawable/buttonselector"/>

    <LinearLayout
        android:id="@+id/BottomButtons"
        android:layout_below="@android:id/list"
        android:layout_width="fill_parent"
```

```
45          android:layout_height="fill_parent"
46          android:orientation="horizontal">
47
48          <ImageButton
49              android:id="@+id/PlayButton"
50              android:src="@drawable/playall"
51              android:background="@drawable/buttonselector"
52              android:padding="5dip"
53              android:layout_marginLeft="40dip"
54              android:adjustViewBounds="true"
55              android:scaleType="fitCenter"
56              android:layout_height="95dip"
57              android:layout_width="95dip"/>
58
59          <ImageButton
60              android:id="@+id/AddButton"
61              android:src="@drawable/add"
62              android:background="@drawable/buttonselector"
63              android:padding="5dip"
64              android:layout_marginLeft="30dip"
65              android:adjustViewBounds="true"
66              android:scaleType="fitCenter"
67              android:layout_height="95dip"
68              android:layout_width="95dip"/>
69
70      </LinearLayout>
71
72  </RelativeLayout>
73
```

storytellingdialog.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="wrap_content"
android:layout_height="250dip"
android:background="#FFFFFF">

    <LinearLayout
    android:id="@+id/Character"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">

        <ImageView
            android:id="@+id/PartChar"
            android:src="@drawable/icon"
            android:adjustViewBounds="true"
            android:scaleType="fitCenter"
            android:layout_margin="20dip"
            android:layout_height="100dip"
            android:layout_width="100dip"/>

        <TextView
            android:id="@+id/PartCharName"
            android:text="Character Name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:textSize = "12sp"
            android:textColor = "#000000"/>
    </LinearLayout>

    <LinearLayout
        android:id="@+id/Participant"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/Character"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/PartParticipant"
            android:src="@drawable/icon"
            android:adjustViewBounds="true"
            android:scaleType="fitCenter"
```

```
45              android:layout_margin="20dip"
46              android:layout_height="100dip"
47              android:layout_width="100dip"/>
48
49          <TextView
50              android:id="@+id/PartParticipantName"
51              android:text="Participant Name"
52              android:layout_width="wrap_content"
53              android:layout_height="wrap_content"
54              android:layout_gravity="center_horizontal"
55              android:textSize = "12sp"
56              android:textColor = "#000000"/>
57      </LinearLayout>
58
59      <Button
60          android:id="@+id/CancelButton"
61          android:text="Stop"
62          android:layout_below="@id/Participant"
63          android:layout_alignRight="@id/Participant"
64          android:layout_width="75dip"
65          android:layout_height="wrap_content"
66          android:layout_margin="20dip"/>
67
68  </RelativeLayout>
```