ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Extensión de plataforma de gestión y fomento de eventos socioculturales mediante lectura de códigos QR

AUTOR: Jorge Vera Arcas
DIRECTOR: Esteban Egea López

Septiembre / 2013

Autor	Jorge Vera Arcas
E-mail del Autor	igekoo@gmail.com
Director(es)	Esteban Egea López
E-mail del Director	esteban.egea@upct.es
Título del PFG	Extensión de plataforma de gestión y fomento de eventos socioculturales, mediante lectura de códigos QR

Resúmen

Hace unos años se desarrolló como PFC una plataforma para la gestión de eventos socioculturales. La plataforma fomentaba la realización de actividades mediante interacciones virtuales en redes sociales, como Facebook.
En su momento se dejó como futura extensión la obtención/carga de eventos en el punto de interés, ya que no estaba tan extendido el acceso a datos mediante móvil.
Hoy en día, el uso de datos en el móvil es habitual y hay disponibles métodos de carga de datos muy intuitivos como son la lectura de códigos de barras bidimensionales.
El objetivo de este PFC es la extensión del proyecto original mediante librerías para la lectura de eventos codificados mediante códigos QR, que se integrarán en la plataforma original.

Titulación	Grado Ingeniería Telemática (Pasarela)
Departamento	Tecnologías de la Información y Comunicaciones
Fecha de Presentación	Septiembre – 2013

AGRADECIMIENTOS

Al Dr. Ing. **Esteban** Egea López

Por la Dirección del Proyecto así como por su apoyo, disponibilidad, documentación ofrecida y medios facilitados para la realización del mismo.

Al Ing. **Antonio Manuel** Padilla Urrea

Por los sabios consejos que me ha ofrecido, siempre a la altura de su conocimiento.

Al Ing. **Francisco** Vera

Por haberme dado la oportunidad y la flexibilidad para que este proyecto y otras muchas cosas salgan adelante.

A mis padres **Juan** y **Paqui** y a mi hermana **Cristina**

Por depositar su confianza en mi y demostrarme su apoyo a lo largo de los años.

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE DE CÓDIGO

CAPÍTULO 1: INTRODUCCIÓN

1.1 Motivación del Proyecto

Hace unos años se desarrolló como PFC una plataforma para la gestión de eventos socioculturales. La plataforma fomentaba la realización de actividades mediante interacciones virtuales en redes sociales, como Facebook.
En su momento se dejó como futura extensión la obtención/carga de eventos en el punto de interés, ya que no estaba tan extendido el acceso a datos mediante móvil.
Hoy en día, el uso de datos en el móvil es habitual y hay disponibles métodos de carga de datos muy intuitivos como son la lectura de códigos de barras bidimensionales.
El objetivo de este PFC es la extensión del proyecto original mediante librerías para la lectura de eventos codificados mediante códigos QR, que se integrarán en la plataforma original.

Hasta ahora, la gran mayoría de los teléfonos móviles estaban orientados a la comunicación por voz. Muchos de estos teléfonos incluían acceso Web pero debido a la escasa facilidad para manejar los navegadores su utilización se veía reducida. Además no todos los terminales disponían de GPS.

La razón de escoger Android como plataforma para la aplicación se justifica por tener todas las características necesarias para esta aplicación; cuenta con conexión 3G, GPS y una pantalla amplia para visualizar con facilidad los mapas de la aplicación. Por lo tanto, todos los terminales de esas características cumplen los requisitos necesarios para poder ejecutar la aplicación.

Otro motivo adicional es el respaldo que ofrece Google Inc. empresa de reconocido prestigio.

Un aspecto interesante a la hora de hacer aplicaciones móviles es la capacidad que ofrece un dispositivo móvil al usuario. Éste ofrece posibilidades similares a las de un ordenador portátil siendo más fácil de llevar consigo.

Con esta tecnología la capacidad de hacer interactuar terminales móviles con el mundo real permite disponer de más información sobre el entorno que nos rodea. Si a esto añadimos la funcionalidad de las redes sociales, obtenemos un valor adicional, donde además de interactuar con el entorno, también se interactúa con los individuos que lo ocupan.

Otro motivo del desarrollo del Proyecto es la versatilidad del mismo: esta aplicación puede ser usada en cualquier parte del planeta y por cualquier tipo de persona.

1.2 Antecedentes

La realidad aumentada (RA) es el término que se usa para definir una visión directa o indirecta de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta en tiempo real. Consiste en un conjunto de dispositivos que añaden información virtual a la información física ya existente. Esta es la principal diferencia con la realidad virtual, puesto que no sustituye la realidad física, sino que sobreimprime información al mundo real. Este proyecto se desarrolla pues en el campo de la realidad virtual, ofreciendo información adicional, sin sobrescribir la realidad.

La realidad aumentada o realidad virtual abarca desde el entorno real al entorno virtual puro. En un punto intermedio está la Realidad Aumentada (más cerca del entorno real) y Virtualidad Aumentada (más cerca del entorno virtual). El caso que nos ocupa está situado en la parte denominada virtualidad aumentada.

Figura 1.1 Realidad real, realidad virtual

1.3 Objetivos del Proyecto

El objetivo de este PFC es la extensión del proyecto original mediante librerías para la lectura de eventos codificados mediante códigos QR, que se integrarán en la plataforma original.

Se extenderá la aplicación móvil para permitir la lectura de eventos codificados con QR. Se extenderá la aplicación servidor para gestionar dichos eventos y generar los códigos apropiados.

Por otra parte, el objetivo genérico de este Proyecto es aprovechar la capacidad de interacción entre el mundo real y el mundo virtual para fomentar el uso y disfrute del

patrimonio natural y cultural. Su fin último consiste en facilitar que una serie de acciones en el espacio virtual se reflejen en el mundo físico y viceversa, haciendo más interesante e interactiva, por ejemplo, una ruta sociocultural.

La realización del Proyecto ha perseguido los siguientes objetivos específicos:

- Aprender a desarrollar aplicaciones para Android en el entorno de desarrollo Eclipse.

- Aprender la dinámica de trabajo a la hora de abordar un proyecto en grupo. Cuadrar ambas partes para su correcto funcionamiento.

- Desarrollar aplicaciones amigables e intuitivas.

- Realizar interfaces gráficas que sean atractivas y fáciles de usar.

- Utilizar herramientas de software libre, para reducir los costes del proyecto.

1.4 Descripción del Entorno

Las siguientes líneas describen el funcionamiento de la herramienta.

La aplicación tiene una función básicamente informativa y sirve como herramienta de planificación turística. El usuario accede a la aplicación Web, consulta los eventos que hay disponibles (rutas turísticas/culturales, rutas de ocio, etc.) y decide a cuál de ellos se quiere suscribir mediante una serie de procedimientos, este último que se ha incorporado es la utilización de los códigos QR como forma simplificada de suscribirse a un evento. Mediante la interfaz gráfica del portal Web los usuarios pueden añadir información adicional con comentarios sobre los itinerarios propuestos con el fin de generar opiniones de los sitios y mejorar la calidad del servicio.

En el caso de que el usuario sea un promotor, por ejemplo el dueño de un negocio o un concejal del Ayuntamiento, puede también crear él mismo nuevos eventos y gestionarlos, además de suscribirse a los que ya hay disponibles. A la hora de crear un nuevo evento, el usuario establece en un mapa los sitios que conforman la ruta que los usuarios apuntados deben realizar. Para que la ruta posteriormente sea validada y completada, se debe recorrer uno a uno todos los puntos del evento. Para que el usuario pueda suscribirse mediante códigos QR será necesario que el promotor utilice una herramienta de generación de dichos códigos para la integración a la hora de suscribirse a los eventos.

Los usuarios se descargan la aplicación para poder realizar las rutas guiándose por su teléfono móvil dotado de GPS. Dicha aplicación es la encargada de ir registrando los sitios por los que va pasando el usuario y de informar cuando se completa una ruta.
El usuario además puede, mediante la aplicación Web, invitar a otros usuarios del sistema a que se suscriban a un evento concreto, mediante la creación de un mensaje de invitación en el que se le remite automáticamente al destinatario el enlace de dicho evento. Por otro lado los usuarios podrán suscribirse a los eventos mediante la captura de los códigos QR generados por los promotores, lo que simplificará tremendamente el proceso de suscripción a eventos nuevos, fomentando así la utilización de la herramienta.

Las oportunidades que ofrece la aplicación son varias. Existe la posibilidad de realizar juegos educativos, como por ejemplo, Gymkhanas creando sitios en las inmediaciones del punto de interés planteando a los usuarios retos relacionados con el paraje. Por otro lado, es posible generar códigos promocionales a los que el usuario solo tenga acceso una vez concluida la ruta. De tal manera que el pequeño o gran negocio puede servirse de la aplicación como herramienta de marketing realizando así campañas para un determinado producto.

La manera de llevar a cabo este propósito es mediante la utilización de dos elementos. Por una parte la herramienta de gestión sería la aplicación Web a la que se accede mediante un navegador Web. Desde aquí es posible crear, editar, suscribirse y compartir los eventos con otros usuarios. Así como interactuar con otros usuarios mediante dos posibles vías:

- Internamente: Enviando solicitudes de amistad a otros usuarios de la aplicación e invitaciones a eventos.
- Externamente: Facilitando, gracias a la integración con Facebook, la recomendación de la aplicación a otros usuarios que puedan estar interesados en su uso. Así como publicar opiniones de otros usurarios con el fin de generar información fiable.

Para entender la mecánica de dicha herramienta es necesario tener en cuenta una serie de conceptos:

Usuarios

- Administrador: usuario con todos los privilegios.
- Promotores: usuarios cuyo fin es fomentar algún tipo de actividad en un determinado emplazamiento creando sus propios eventos.
- Usuarios: solo tienen la posibilidad de apuntarse a los eventos y compartirlos con sus amigos. Su propósito es seleccionar los eventos que más le interesan para realizar algún tipo de actividad ya sea cultural, turística o de ocio.

Sitios: Serie de puntos geoposicionados que indican la ubicación de algún enclave de interés para los usuarios.

Dispositivos: Son terminales que se colocan en determinadas ubicaciones para mejorar el funcionamiento del sistema. Cada dispositivo permite el acceso mediante WIFI a datos

específicos sobre el entorno en el que está colocado. En esta versión de la aplicación no esta todavía implementado.

Eventos

- Ruta / Gymkhana: serie de sitios ubicados específicamente en enclaves con interés turístico o cultural que aportan al usuario información adicional sobre el entorno.
- Código promocional: es una ruta con la particularidad de que al acabarla se premia al usuario con un código canjeable por alguna recompensa como por ejemplo descuentos, obsequios, etc…

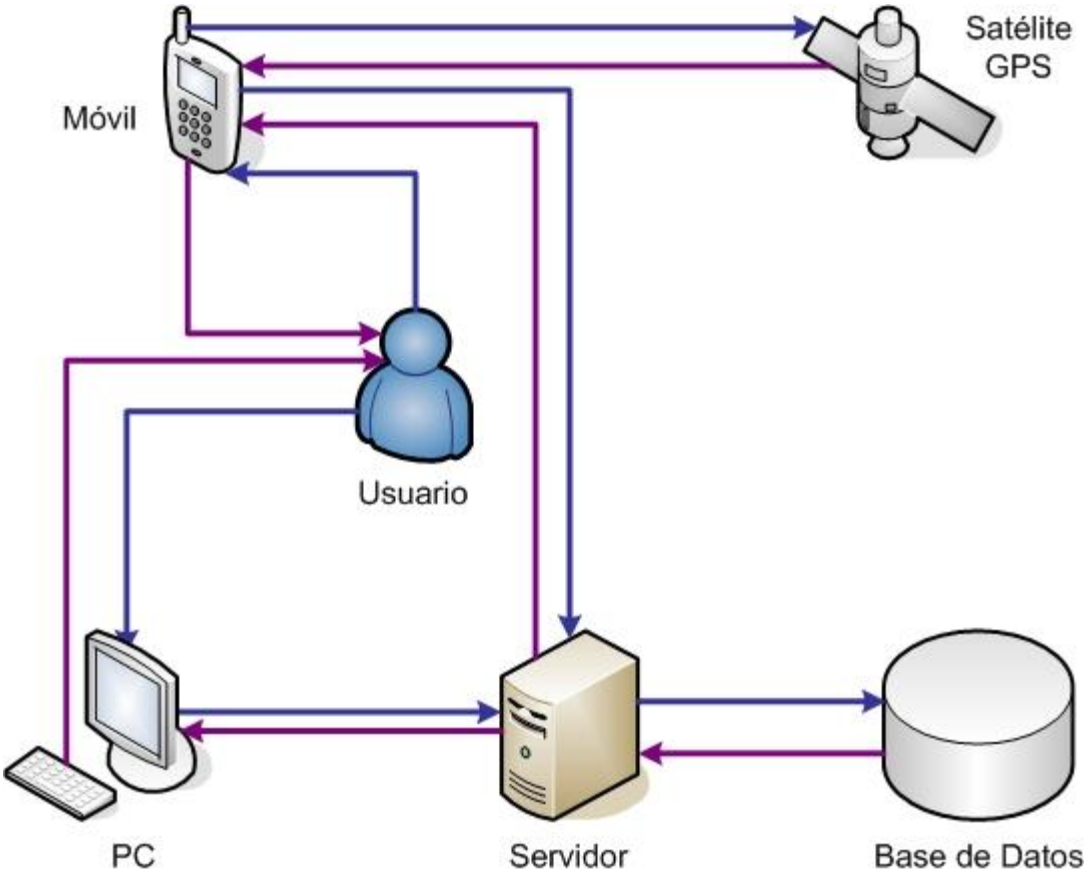


Figura 1.2 Entorno de la Aplicación

1.5 Estructura de la Memoria

Capitulo 1: Introducción
Capitulo 2: Tecnologías empleadas
Capitulo 3: Desarrollo del proyecto
Capitulo 4: Conclusiones y líneas futuras
Capitulo 5: Referencias – Bibliografía
Capitulo 6: Anexo

CAPÍTULO 2: TECNOLOGÍAS EMPLEADAS

Se muestra a continuación la figura que describe el cometido de cada una de las tecnologías empleadas para el funcionamiento del proyecto global siendo usadas, para este proyecto específicamente las siguientes: Android, Eclipse, Apache, MySQL, PhpMyAdmin y PHP.

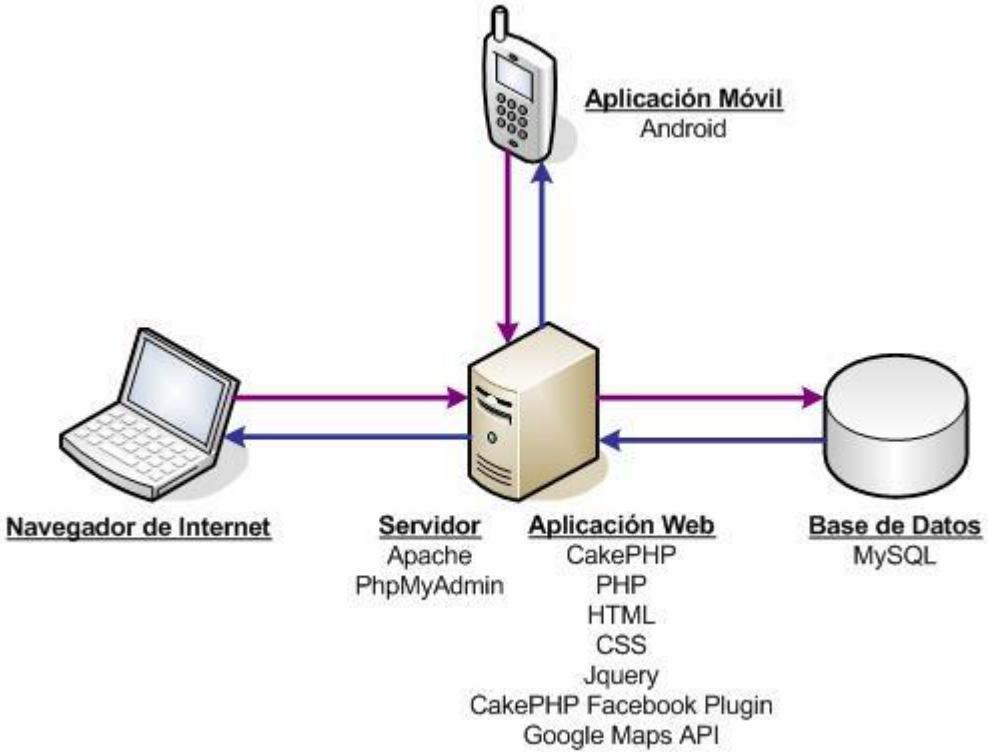


Figura 2.1 Tecnologías empleadas.

2.1 Plataforma Android

2.1.1 Historia

El lanzamiento inicial del Android Software Development Kit (SDK) apareció en noviembre de 2007 y a mediados de agosto de 2008 apareció el Android 0.9 SDK en version beta. Al mes siguiente lanzaron Android 1.0 SDK (Release 1).

A principios de marzo 2009, Google presentó la versión 1.1 de Android para el "dev phone" y la actualización incluía algunos cambios estéticos además de soporte para "búsquedas por voz", la posibilidad de comprar aplicaciones en Android Market, arreglos en el reloj alarma, mejoras en Gmail y demás.

A mediados de mayo 2009, Google lanza la versión 1.5 de Android OS (llamada Cupcake) con su respectivo SDK que incluía nuevas caracteristicas como: grabación de video, soporte para stereo Bluetooth, sistema de teclado personalizable en pantalla, reconocimiento de voz y el AppWidget framework que permitió que los desarrolladores puedan crear sus propios widgets para la página principal. Android 1.5 fue la versión que más personas usaron para iniciarse en Android (con el T-Mobile G1 y HTC Dream en USA) y sigue siendo actualmente una versión que se encuentra disponible en muchos móviles Android como el HTC Hero. Móvil que se ha usado para implementar la aplicación.

Para continuar con la evolución de Android, el Google Nexus One llegó con la versión Android 2.1, el cual marcó un antes y un después ya que Google trató de venderlo por su cuenta y liberado. A esta versión algunos llamaron "Flan" pero Google sigue considerándolo parte de "Eclair" porque trae nuevas capacidades 3D, live wallpapers, lo que significó la gran mejora de la plataforma desde la versión 1.6.

La última versión de Android fue anunciada el 20 de mayo del 2010 y está disponible desde finales de junio del 2010, el cual ha sido denominada Android 2.2 "FroYo" (Frozen Yogurt).

2.1.2 Arquitectura

La arquitectura de Android fomenta el concepto de reutilización de componentes, lo que le permite publicar y compartir Actividades, Servicios y datos con otras aplicaciones, con acceso controlado por las restricciones de seguridad que se apliquen. El mismo

mecanismo que le permite producir un marcador telefónico, puede dejar que se expongan los componentes de aplicación, que permite a los desarrolladores crear una nueva interfaz de usuario para otros fines y extensiones de sus funcionalidades, o de otro modo construir sobre ellos.

La figura 2.1, muestra el sistema de arquitectura jerárquico de las aplicaciones Android que contiene los bloques que posee cada capa de éste sistema, a continuación se detallará sobre los diferentes componentes principales:

Núcleo (kernel) de Linux: Los servicios básicos (incluidos los controladores de hardware, procesos y gestión de memoria, seguridad, red y administración de energía) son manejados por un núcleo Linux 2.6. El núcleo también provee una capa de abstracción entre el hardware y el resto de la pila.

Librerías: Corriendo en la parte superior del núcleo, Android incluye varias librerías base de C/C++ (como: libc y Secure Sockets Layer, SSL), así como también:

- Librerías para reproducción de audio y video.

- Un administrador de superficies para proveer gestión de pantalla.

- Librerías graficas que incluyen SGL y OpenGL para gráficos 2D y 3D.

- SQLite para soporte de la base de datos nativa.

- SSL y WebKit para la integración de navegador web y seguridad en internet.

Runtime de Android: Android incluye un grupo de librerías base que proveen la mayor parte de las funcionalidades disponibles en las librerías base del lenguaje de programación Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Esta máquina ejecuta archivos en el formato Dalvik Ejecutable (.dex), el cual está optimizado para uso de memoria mínima.

Framework de aplicaciones: El framework de aplicaciones provee las clases usadas para crear aplicaciones Android. Además de permitir abstracciones genéricas de acceso al hardware y administración de las interfaces de usuario y recursos de la aplicación.

Capa de Aplicación: Todas las aplicaciones, las nativas y de terceros, son construidas en esta capa utilizando las mismas librerías API. Esta capa corre en tiempo de ejecución por medio del uso de las clases y servicios disponibles

para el framework de aplicaciones.

A continuación se va a profundizar un poco mas en la plataforma Android ya que hay conceptos que son de gran importancia para comprender la utilidad de esta plataforma para el desarrollo de aplicaciones móviles.

2.1.3 Características

Dentro de las principales características de Android destacan:

- Framework de aplicaciones: Permite reutilización y reemplazo de componentes.

- Máquina virtual Dalvik: Optimizada para dispositivos móviles.

- Navegador integrado: Basado en el motor de código abierto WebKit.

- Gráficos optimizados: Con una biblioteca de gráficos 2D; gráficos 3D basado en la especificación OpenGL ES 1.0 (aceleración por hardware opcional).

- SQLite: Para almacenamiento de datos estructurados.

- Soporte para medios: Con formatos comunes de audio, vídeo e imágenes planas (MPEG4, H.264, MP3, OGG, AAC, AMR, JPG, PNG, GIF).

- Telefonía GSM: dependiente del hardware.

- Bluetooth, EDGE, 3G, y WiFi (dependiente del hardware).

- Cámara, GPS, brújula, y acelerómetro (dependiente del hardware).

- Ambiente amigable de desarrollo: incluyendo un emulador de dispositivo, herramientas para depurar, perfiles de memoria y rendimiento, y un complemento para el IDE Eclipse.

- Pantalla táctil.

- Android Market: Permite que los desarrolladores pongan sus aplicaciones, gratuitas o de pago, en el mercado a través de esta aplicación accesible desde la mayoría de los teléfonos con Android.

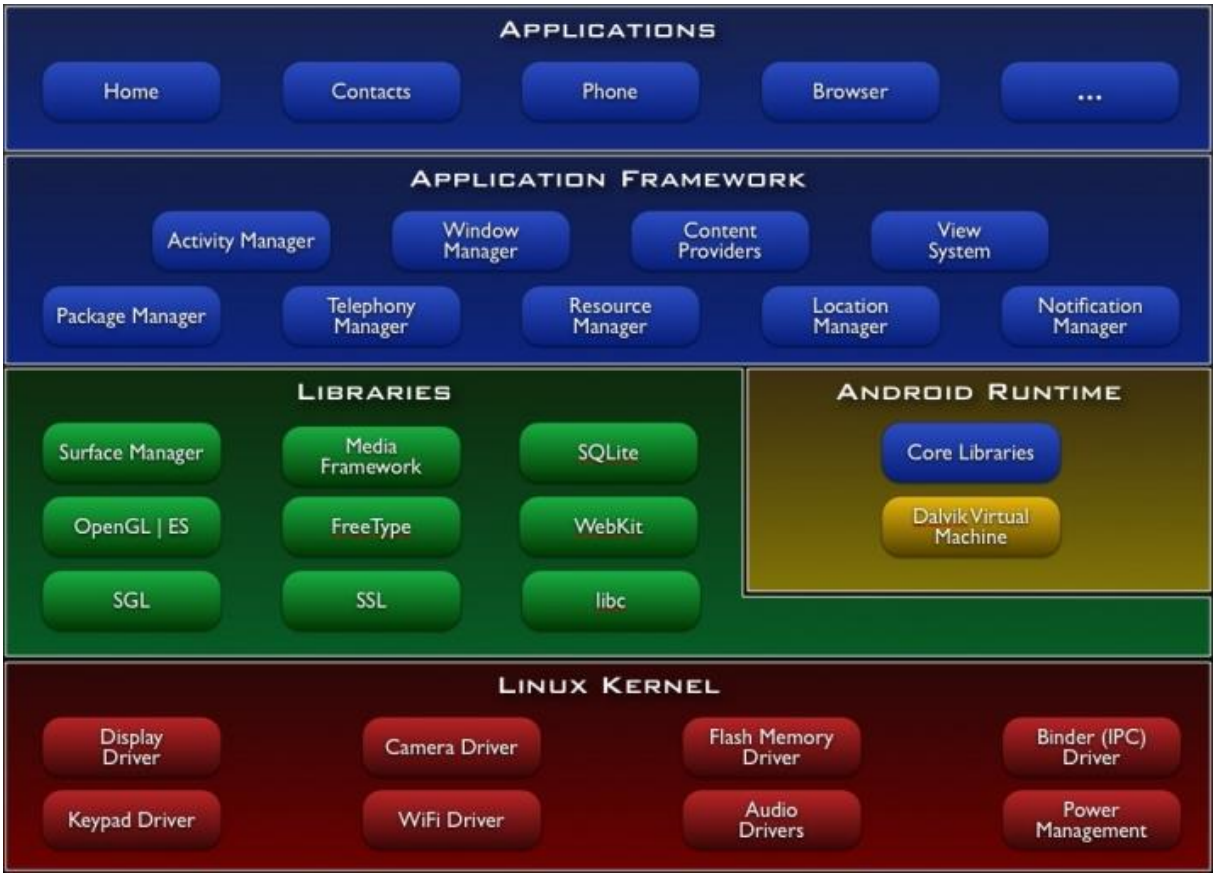


Figura 2.2. Arquitectura Android

2.1.4 Framework

Al proporcionar una plataforma de desarrollo abierto, Android ofrece a los programadores la capacidad de crear aplicaciones muy ricas e innovadoras. Los desarrolladores pueden aprovechar las ventajas que les ofrece el hardware del dispositivo, información sobre la ubicación de acceso, ejecutar servicios de fondo, establecer alarmas, añadir las notificaciones de la barra de estado, y un sinfín de ventajas más.

Los desarrolladores tienen pleno acceso a las API teniendo el mismo marco utilizado por las aplicaciones principales. La arquitectura de la aplicación está diseñada para simplificar la reutilización de componentes, de tal manera que cualquier aplicación puede publicar sus capacidades permitiendo a otras aplicaciones hacer uso de esas capacidades (sujeto a restricciones de seguridad impuestas por el marco).

Englobando todas las aplicaciones se tiene un conjunto de servicios y sistemas que incluyen:

- Un amplio conjunto de vistas que pueden utilizarse para crear una aplicación, incluidas las listas, redes, cajas de texto, botones, además de un navegador Web embebido.

- Los proveedores de contenido que permiten a las aplicaciones de acceso a datos de otras aplicaciones (por ejemplo, Contactos), o para compartir sus propios datos.

- Un administrador de recursos, facilitar el acceso a los recursos no son de código como cadenas localizadas, gráficos y archivos de diseño.

- Un Administrador de notificaciones que permite a todas las aplicaciones para mostrar alertas personalizadas en la barra de estado.

- Un gestor de actividad que gestiona el ciclo de vida de las aplicaciones y proporciona una navegación backstack común.

2.1.5 Componentes de una aplicación

Todas las aplicaciones en Android pueden descomponerse en cuatro tipos de bloques o componentes principales. Cada aplicación será una combinación de uno o más de estos componentes. Estos deberán ser declarados de forma explícita en un archivo con formato XML denominado “AndroidManifest.xml”, junto a otros datos asociados como valores globales, clases que implementa, datos que puede manejar, permisos, etc. Este archivo es básico en cualquier aplicación en Android y permite al sistema desplegar y ejecutar correctamente la aplicación.

A continuación se exponen los cuatro tipos de componentes en los que puede dividirse una aplicación Android:

Activity - Actividad

Es el componente que más se utiliza en el desarrollo de las aplicaciones para Android. Un componente Activity muestra una determinada actividad llevada a cabo por una aplicación, que normalmente tiene asociada una ventana o interfaz de usuario. Es muy importante señalar que este componente no contempla únicamente el aspecto gráfico, sino que éste forma parte del componente Activity a través de vistas representadas por clases como View y sus derivadas. Se implementa mediante la clase de mismo nombre Activity.

Gran parte de las aplicaciones permiten la ejecución de varias acciones a través de la

existencia de una o más pantallas. Por ejemplo, pensemos en una Aplicación para el envió de mensajes de texto: en ella, la lista de contactos se muestra en una ventana. Mediante el despliegue de una segunda ventana, el usuario puede escribir el mensaje al contacto elegido, y en otra puede repasar su historial de mensajes enviados o recibidos. Cada una de estas ventanas debería estar representada a través de un componente Activity, de forma que navegar de una ventana a otra implica lanzar una actividad o dormir otra. Android permite el manejo por completo el ciclo de vida de los componentes Activity.

Los Intents son una interesante novedad introducida por Android, el cual se encuentran muy vinculados a este componente. Un Intent consiste básicamente en la ejecución de alguna acción, que generalmente está asociada a unos datos. Lanzando un Intent, una aplicación puede delegar el trabajo en otra, de forma que el sistema se encarga de buscar qué aplicación entre las instaladas es la que puede llevar a cabo la acción solicitada. Por ejemplo, abrir una dirección URL en algún navegador web, o escribir un correo electrónico desde algún cliente de correo.

BroadcastReceiver – Receptor de broadcast

Se utiliza para lanzar alguna ejecución dentro de la aplicación actual cuando un determinado evento se produzca, que por lo general es abrir un componente Activity. Por ejemplo, una llamada entrante o un SMS recibido. No tiene interfaz de usuario asociada, pero puede utilizar el API Notification Manager, mencionada anteriormente, para avisar al usuario del evento producido a través de la barra de estado del dispositivo móvil. Este componente se implementa a través de una clase de nombre BroadcastReceiver.

Para que un BroadcastReceiver funcione, no se necesita que la aplicación en cuestión sea la aplicación activa en el momento de producirse el evento. El sistema lanzará la aplicación si es necesario cuando el evento monitorizado tenga lugar.

Service - Servicios

Es aquel que representa una aplicación ejecutada sin interfaz de usuario, y que generalmente tiene lugar en segundo plano (background) mientras otras aplicaciones (éstas con interfaz) son las que están activas en la pantalla del dispositivo. Un ejemplo muy común es un reproductor de música. La interfaz del reproductor muestra al usuario las distintas canciones disponibles, así como los botones de reproducción, pausa, volumen, etc. En el momento en el que el usuario reproduce una canción, ésta se escucha mientras se siguen operando en todas las acciones anteriores, e incluso se puede ejecutar una aplicación distinta sin que la música deje de sonar. La interfaz de usuario del reproductor sería un componente Activity, pero la música en reproducción sería un componente Service, porque se ejecuta en background. Este elemento está implementado por la clase de mismo nombre Service.

ContentProvider – Proveedor de contenido

Se utiliza para almacenar datos en un archivo, en una base de datos SQLite o en cualquier otro formato que considere para que cualquier aplicación pueda utilizarlo. Además, estos datos pueden ser compartidos entre distintas aplicaciones. Una clase que implemente el componente ContentProvider contendrá una serie de métodos que permite almacenar, recuperar, actualizar y compartir los datos de una aplicación.

Existe una colección de clases para distintos tipos de gestión de datos en el paquete android.provider. Además, cualquier formato adicional que se quiera implementar deberá pertenecer a este paquete y seguir sus estándares de funcionamiento.

2.1.6 Ciclo de vida de una aplicación Android

En muchos casos, una aplicación Android corre dentro de su propio proceso Linux. Este proceso es creado para la aplicación cuando parte de su código necesita ser ejecutado, y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

Una característica importante, y poco usual, de Android es que el tiempo de vida de un proceso no es controlado directamente por la aplicación. En lugar de esto, es el sistema quien determina el tiempo de vida del proceso basado en el conocimiento que tiene el sistema de las partes de la aplicación que están corriendo, la importancia de la aplicación para el usuario y cuánta memoria disponible hay en un determinado momento.

Es muy importante tener en consideración cómo los distintos componentes de una aplicación (en particular "Activity", "Service" y "IntentReceiver") impactan en el tiempo de vida del proceso asociado con una aplicación. Si estos componentes no son empleados de manera apropiada, el sistema detendrá el proceso de la aplicación aún cuando se esté haciendo algo importante.

Como caso práctico, un ejemplo de lo anterior sería un "IntentReceiver" que inicia un "thread" cuando recibe un "Intent" en su método "onReceiveIntent()" y entonces finaliza el método y devuelve el control. Una vez que el método devuelve el control, el sistema considera que el "IntentReceiver" no continúa activo y por lo tanto el proceso asociado a él ya no se requiere mantener corriendo (a menos que otro componente de la aplicación esté activo dentro de él). Debido a esto, el sistema en cualquier momento que requiera memoria eliminará el proceso y como consecuencia también el "thread" que fue creado anteriormente. La solución a este problema es iniciar un "Service" desde el "IntentReceiver", de esa manera el sistema sabrá que después de que "IntentReceiver" finalice aún habrá un trabajo activo asociado al proceso.

Para determinar qué procesos deberían ser eliminados ante una condición de baja

memoria, Android ordena los procesos en una estructura jerárquica y asignándole a cada proceso una determinada importancia basada en los componentes que están corriendo dentro de ellos y el estado de aquellos componentes.

De acuerdo a esta jerarquía de importancia se distinguen los siguientes tipos de procesos:

Foreground process: (proceso de primer plano) es un proceso que hospeda una actividad en la pantalla y con la cual el usuario interactúa (su método "onResume()" ha sido llamado) o un IntentReceiver está corriendo (su método "onReceiveIntent()" se está ejecutando). Estos procesos sólo serán eliminados cómo último recurso, si es que la memoria está tan baja que ni siquiera estos procesos pueden continuar corriendo. De ser asi, el dispositivo alcanzará un "memory paging state" y esta acción es necesaria para mantener la interfaz del usuario con capacidad para responder gráficamente.

Visible process: (proceso visible) es un proceso que hospeda una "Activity" que está visible en la pantalla, pero no en el primer plano (su método "onPause()" ha sido llamado). Esto podría ocurrir, si por ejemplo, la actividad de primer plano está siendo desplegada en la pantalla con la apariencia de un cuadro de diálogo que permite que la actividad previa pueda ser vista detrás de ella. Este tipo de proceso es considerado extremadamente importante y no será eliminado a menos sea estrictamente necesario para mantener a todos los procesos de primer plano corriendo.

Service process: (proceso de servicio) es un proceso que hospeda a un "Service" que ha sido inicializado con el método "startService()". Aunque estos procesos no son directamente visibles al usuario, generalmente están haciendo tareas que para el usuario son muy importantes (tales como reproducir un archivo mp3 o mantener una conexión con un servidor de contenidos), por lo tanto el sistema siempre tratará de mantener esos procesos corriendo a menos que los niveles de memoria comiencen a comprometer el funcionamiento de los procesos de primer plano.

Background process: (proceso de fondo) es un proceso que hospeda una "Activity" que no es actualmente visible al usuario (su método "onStop()" ha sido llamado). Si estos procesos son eliminados no tendrán un directo impacto en la experiencia del usuario. Basado en el hecho que la "Activity" ha sido programada para implementar correctamente su ciclo de vida, el sistema puede eliminar estos procesos en cualquier momento para reclamar la memoria que estén usando para entregársela a cualquiera de los tres tipos de procesos anteriormente descritos. Generalmente, hay muchos de estos procesos corriendo, por lo tanto el sistema mantiene una lista "LRU" para asegurar que el último proceso visto por el usuario sea el último en ser eliminado en caso que se requiere recolectar memoria.

Empty process: (proceso vacío) es un proceso que no hospeda a ningún componente de

aplicación activo. La única razón para mantener ese proceso es tener una caché que permita mejorar el tiempo de activación en la próxima oportunidad que un componente de su aplicación requiera correr. Como consecuencia de esto, el sistema con frecuencia va a eliminar estos procesos para mantener el balance entre los recursos de la memoria caché utilizados por estos procesos con los utilizados por los caché del kernel del sistema.

Al momento de clasificar un proceso, el sistema toma el nivel más importante de todos los componentes activos de un proceso. Lee sobre la documentación de "Activity", "Service" y "IntentReceiver" para estudiar más detalles de cómo cada uno de estos componentes contribuyen en el ciclo de vida del proceso. La documentación de cada una de estas clases también entregan más detalles de cómo estos componentes impactan en el ciclo de vida de una aplicación.

Por lo tanto el punto de entrada de una aplicación normalmente será una actividad (es decir, la pantalla que vemos cuando ejecutamos la aplicación). Ésta actividad se compone de dos elementos: la interfaz (que estará en res/layout) y el código fuente (que estará en src).

Una actividad tiene un ciclo de vida muy definido, heredado de los objetos padre del SDK, y que será igual para todas las actividades. Algunos de los métodos del ciclo de vida son los siguientes:

- **void onCreate(Bundle savedInstanceState):** Creación de la Actividad. Puede recibir información de estado de instancia, por si se reanuda desde una instancia pausada anterior.

- **void onStart()**: Inicio de la actividad recién creada.

- **void onRestart()**: Reinicio de una actividad tras una pausa.

- **void onPause()**: Método ejecutado antes de pausar la aplicación

- **void onStop()**: Fin de la actividad.

- **void onDestroy()**: Método ejecutado antes de destruir del todo la actividad.

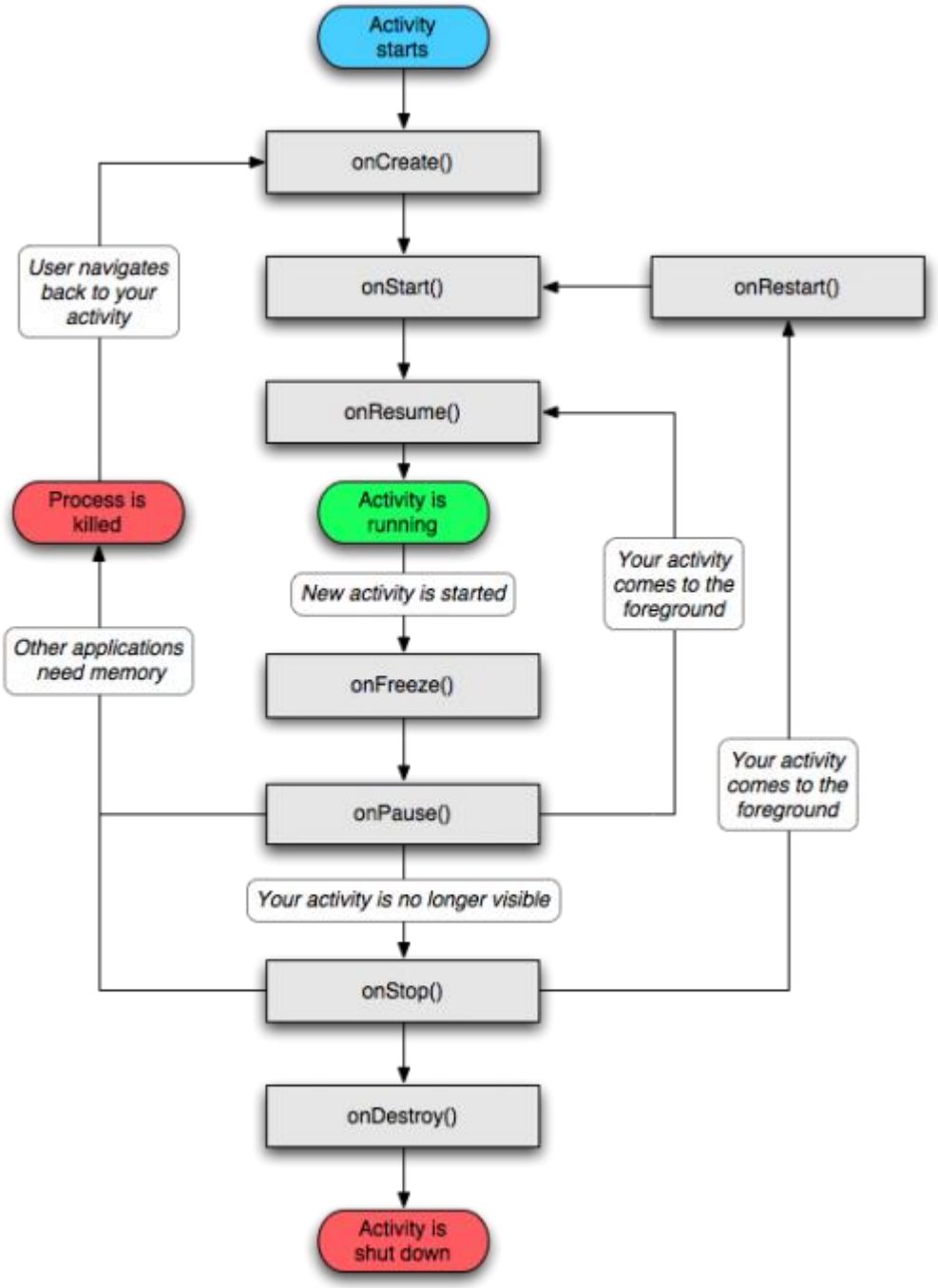


Figura 2.3 Ciclo de vida de una aplicación Android.

2.1.7 API Demos

Como parte del SDK de Android, Google ha incluido un gran número de clases dentro de un bloque denominado API Demos. Estas clases muestran de forma práctica muchas de las características y posibilidades ofrecidas por el sistema Android. Algunas de ellas son las siguientes:

App: Incluye ejemplos de cómo usar los principales componentes de Android, como las *Activity* y *Service*, además de otras aplicaciones sencillas de muestras. En estas clases se pueden encontrar ilustraciones sobre cómo usar las preferencias de usuario, devolver un valor desde otra actividad, lanzar componentes *Service* en el mismo o distinto proceso, o sobre cómo lanzar alarmas y notificaciones al usuario.

View: Aquí se encuentran ejemplos de distintos tipos de vistas y diseños para las interfaces de usuario. Además se encuentran clases que ilustran la creación de formularios, distribuciones verticales y horizontales de elementos, vistas que se adaptan al contenido, ventanas con *scroll*, listados de elementos, botones compuestos por imágenes, galerías de imágenes o barras de progreso.

Graphics: Se muestran ejemplos de la capacidad gráfica de Android, tanto de la definición de imágenes con movimientos como de la construcción de figuras geométricas básicas.

Media: Incluye ejemplos sobre el manejo de formatos multimedia, como: sonidos, vídeo e imágenes.

Estos códigos de ejemplo suelen ser muy útiles, cuando se conoce tanto el funcionamiento más básico de Android, como sus características principales, y resultan de gran provecho para comenzar a hacerse una idea general del tipo de aplicaciones que este sistema puede soportar.

2.1.9 Estructura de un proyecto Android

Cuando se crea un proyecto Android en Eclipse, se generan una serie de carpetas con sus correspondientes subcarpetas que forman el proyecto. Estas carpetas y subcarpetas se crean automáticamente gracias al plug-in ADT de eclipse y constituyen la estructura en la que se va a ayudar el programador y donde va a disponer de una organización estándar.

Figura 2.4 Estructura de la aplicación.

Cada uno de los principales elementos de la aplicación tiene un propósito distinto que se va a detallar a continuación:

2.1.8.1<Android Version>

Indica la versión del proyecto como por ejemplo, en este caso la Android 1.5.
Incluye el android.jar, archivo que tu aplicación ira creando cada vez que se compile el programa. Este archivo viene determinado por la elección realizada al crear el proyecto.

2.1.8.2Carpeta \src

Esta carpeta contiene los archivos fuente del proyecto. Está organizada según el paquete donde se encuentren los archivos .java. Es decir si el paquete se llama com.prueba los archivos iran dentro de la carpeta src/com.prueba.
El nombre de la actividad principal vendrá indicado por el nombre del proyecto con la extensión .java.

2.1.8.3 Carpeta \Gen

Gen de Generated Java Files, archivos java autogenerador. Esta carpeta contiene el fichero, "R.java", es un archivo que siempre se adjunta por defecto a cualquier proyecto Android y que declara una serie de índices o referencias a los recursos externos que se utilizan en el proyecto actual.

2.1.8.4 Carpeta \assets

Esta carpeta está vacía y se puede usar para almacenar datos posteriormente.

2.1.8.5 Carpeta \res

Esta carpeta contiene los recursos que se van a utilizar en el proyecto. Un recurso es cualquier fichero externo que contenga datos y/o descripciones referentes a la aplicación. Estos deben ser compilados junto con la aplicación ya que sin ellos no funcionaria correctamente. Gracias a esta compilación la aplicación accede de manera más rápida y sencilla a estos recursos que pueden presentarse en numerosos formatos tales como XML para texto y JPEG ó PNG para imágenes entre otros muchos.

En concreto para realizar un proyecto en Android los recursos son muy utilizados como fichero XML donde se describen y configuran muchos de los elementos gráficos de un proyecto Android por ejemplo. Otro caso muy útil de un recurso en Android es la facilidad de implementar traducciones de una aplicación a distintos idiomas.

Por lo tanto, cada uno de los recursos de una aplicación para Android ha de estar referenciado en la subcarpeta adecuada según el tipo de referencia que sea. Las diferentes subcarpetas para organizar los recursos son:

\drawable

Esta carpeta contiene los recursos que pueden ser dibujados en la pantalla como por ejemplo imágenes JPG PNG ó GIF

\layout

Aquí se almacenan los layouts o diseños que más tarde serán utilizados para construir las interfaces gráficas. Permitiendo configurar según las necesidades de la aplicación la pantalla. Por convención nada mas crear un proyecto de Android el Asistente creará un archivo denominado main.xml donde se guardará la vista de la pantalla principal de nuestro nuevo programa

\values

En esta carpeta es donde se guardan los ficheros XML que declaran los valores que queramos definir, como por ejemplo, cadenas de texto. La manera de acceder a estos recursos con posterioridad será desde el código a un archivo denominado strings.xml que al igual que el main.xml será creado automáticamente por el Wizard nada más crear el proyecto

Figura 2.5 Jerarquía de la carpeta "res"

Todos los recursos declarados en estos archivos quedaran reflejados en el fichero fuente R.java que contiene una subclase por cada tipo distinto de recurso que se haya declarado que a su vez declaran una serie identificadores que representan cada uno de los recursos declarados. Gracias a este fichero, los recursos se pueden utilizar y referenciar directamente en el código de la aplicación mediante el método getResources() de la clase Context

Este fichero se sincroniza y genera automáticamente en tiempo de compilación, por lo que no se aconseja modificarlo manualmente.

2.1.8.6 Carpeta \bin

Esta carpeta contiene los archivos binarios del proyecto generados a partir de los archivos fuente. Al igual que la carpeta "\src", se mantiene la misma jerarquía de subcarpetas que la representada en el nombre del paquete.

2.1.8.7 default.properties

Este archivo contiene la configuración del proyecto como el Build Target. Este archivo nunca debería ser modificado manualmente.

2.1.8.8 AndroidManifest.xml

Un elemento básico de una aplicación Android es el archivo denominado AndroidManifest.xml. Este archivo se genera de manera automática por el plug-in de Eclipse al crear el proyecto y será modificado según las necesidades del proyecto. Su localización en la carpeta del proyecto es el directorio raíz y viene representada por la Figura 2.6:

Figura 2.6 Localización del archivo AndroidManifest.xml

Sirve para describir el contexto de la aplicación y sus actividades, servicios, receptores y /o proveedores de contenido además de otro tipo de permisos. Las utilidades de AndroidManifest.xml son:

- Nombrar el package de la aplicación. Este sirve como identificador único de la aplicación.

- Describir los componentes de la aplicación: el contexto de la aplicación, sus actividades, servicios, receptores y /o proveedores de contenido. Nombra cada una de las clases que componen las actividades.

- Determinar que procesos albergarán los componentes de la aplicación

- Declarar que permisos tendrá la aplicación para así proteger partes de la API e interactuar con otras aplicaciones.

- También se declaran los permisos que otros tienen con respecto a los componentes de la aplicación.

-Declara el nivel mínimo requerido de la API de Android.

- Enumera las clases de instrumentación que proporcionan información mientras la aplicación se está ejecutando. Estas declaraciones están presentes en el manifiesto sólo cuando la aplicación está siendo desarrollada y probada, luego se eliminaran antes de publicar la aplicación.

- En el se enumeran las clases de instrumentación que proporcionan la información de los perfiles y datos de la aplicación que se esta ejecutando. Este tipo de declaración solo esta presente en el AndroidManifest.xml cuando la aplicación esta en fase de desarrollo y pruebas. Son eliminados cuando se pública.

- Lista las librerías que necesita la aplicación, por ejemplo en nuestro caso los mapas de Google.

A modo de ejemplo se va a mostrar el aspecto del archivo final AndroidManifest.xml del proyecto es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.login"
      android:versionCode="1"
      android:versionName="1.0">
      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
      <uses-permission android:name="android.permission.INTERNET" />
    <application android:icon="@drawable/icon" android:label="@string/app_name" android:debuggable="true">
        <uses-library android:name="com.google.android.maps" />
        <activity android:name=".Login"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    <activity android:name="Actividad" android:label="@string/app_name">
    <intent-filter>
    </intent-filter>
    </activity>
```

Figura 2.7 AndroidManifest.xml final del proyecto.

Por lo tanto, la manera de estructurar un archivo AndroidManifest.xml se realiza siguiendo las normas y elementos que se describen en la siguiente tabla:

Elemento	Posición	Descripción
<manifest>	Raíz	Define el paquete de la aplicación y el espacio de nombres de Android.
<uses-permission>	Raíz	Solicita un permiso de seguridad.
<permision>	Raíz	Declara un permiso de seguridad.
<application>	Raíz	Define una aplicación nombre de clase, etiqueta, icono o tema.
<activity>	Nodo hijo de <application>	Define una clase Activity.
<intent-filter>	Nodo hijo de <activity>	Declara los Intents admitidos por Activity.
<action>	Nodo hijo de <intent-filter>	Acción de Intent.
<category>	Nodo hijo de <intent-filter>	Categoría de Intent.

Tabla 2.1 Estructura de AndroidManifest.xml

2.1.9 Emulador

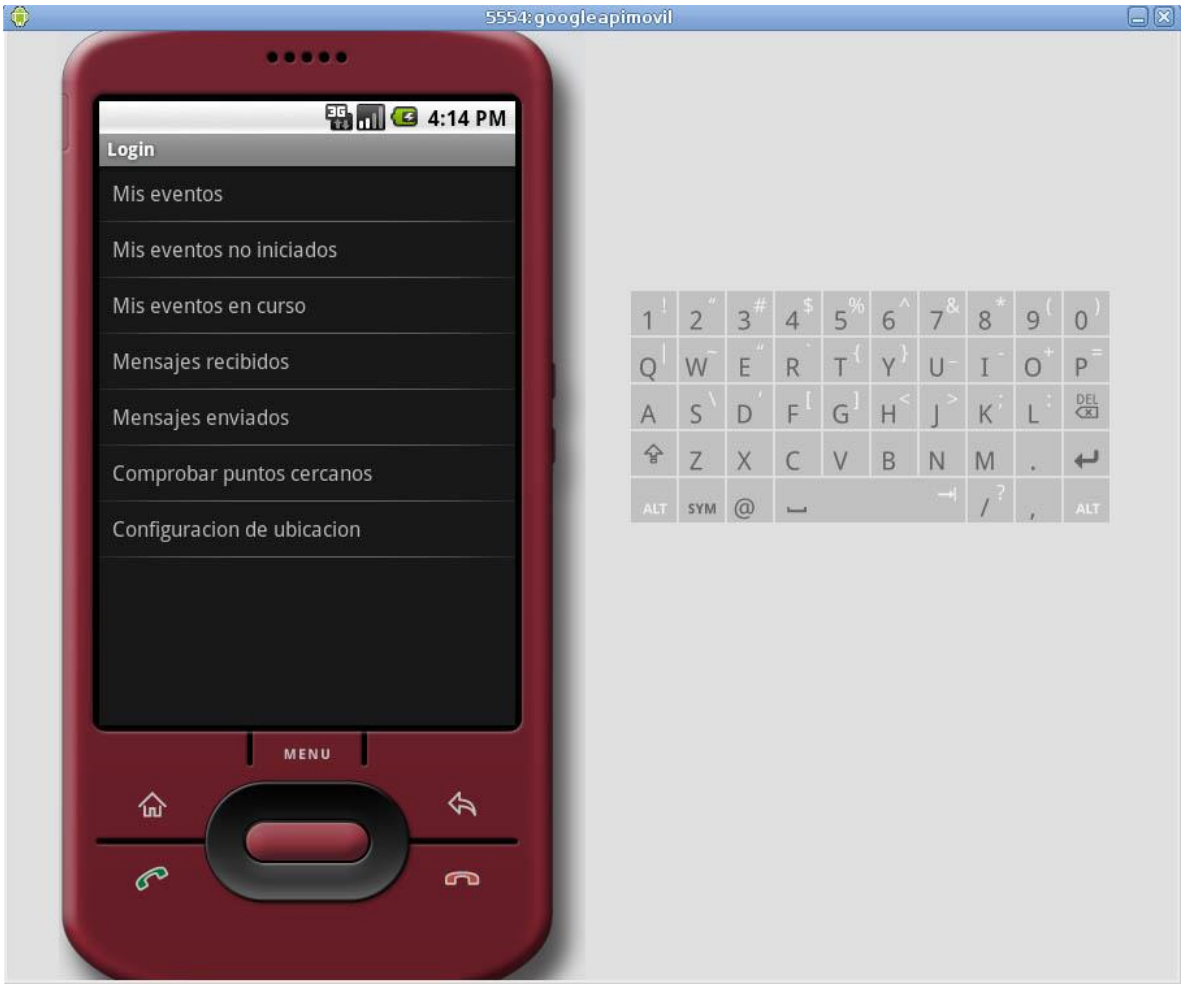


Figura 2.8 Ejemplo de skin para el emulador de Android.

Para hacer las pruebas y la depuración del código implementado el SDK de Android incluye un emulador. Es una de las herramientas más útiles y potentes a la hora de desarrollar aplicaciones para Android es el emulador. El emulador consta de varios skins que permiten modificar la apariencia grafica, del dispositivo a emular, botones y pantalla además de las aplicaciones incluidas en un móvil real.

Esta herramienta es lanzada desde un plug-in desde Eclipse aunque también puede ser ejecutada desde una ventana de comandos accediendo a la carpeta /tools y ejecutando el archivo emulator.exe

El caso del plug-in de Eclipse, que es precisamente el que se ha utilizado para el desarrollo del proyecto, la ejecución del emulador se hace automáticamente.
Esta herramienta se puede configurar en Run/Run configurations -> Target
Cuando se lanza el emulador puede interactuarse con éste a través de su interfaz gráfica pulsando en la pantalla del dispositivo, sobre sus botones de función o en el teclado virtual. Permitiendo simular llamadas entrantes, mensajes SMS y desde la consola configurar ubicaciones.

Órdenes desde consola

Aunque el emulador ya esté funcionando, hay muchos aspectos de este que pueden ser configurados utilizando la consola a través de Telnet al puerto 5554 y posteriores, según la instancia del emulador deseada.

Algunos de los comandos aceptados por la consola son los siguientes:

- **help**: muestra una lista de los comandos disponibles.

- **event**: permite enviar diferentes tipos de eventos al emulador.

- **geo**: permite establecer una localización fija mediante coordenadas. Será la que devuelva el dispositivo de GPS al ser invocado desde una aplicación.

- **gsm**: emula llamadas telefónicas.

- **kill**: finaliza la instancia del emulador.

- **network**: diferentes aspectos relacionados con el control de los distintos tipos de conexiones (GSM, GPRS, UMTS, EDGE), como los valores de latencia.

- **power**: controla la simulación del nivel de batería del dispositivo móvil.

- **quit/exit**: cierra la conexión con el emulador y la consola.

- **redir**: permite redireccionar puertos entre el emulador y la máquina de desarrollo.

- **sms**: permite enviar mensajes SMS al emulador.

- **vm**: control de la máquina virtual Dalvik.

- **window**: configuración del skin del emulador.

2.1.10 Dalvik Debug Monitor

La herramienta Dalvik Debug Monitor está fuertemente ligada al emulador ya que es otro elemento de testeo de aplicaciones. Permite ver información sobre los procesos en ejecución, gastos de memoria, eventos y sobre todo, lo que mas se ha utilizado en este proyecto los mensajes de log. Estos mensajes son muy útiles a la hora de depurar la aplicación ya que nos permite ver hasta que punto de la aplicación llega la ejecución para arreglar errores de implementación.

Como se ha indicado anteriormente en este apartado es una herramienta fuertemente ligada al emulador ya que para utilizarla éste ha de estar en ejecución. Al saltar la ventana del emulador, solo hay que pinchar sobre una ventana llamada DDMS (o en la ventana de comandos ejecutar ddms.exe) en la ventana de eclipse y cambiará a la vista del Debug Monitor. (Figura 2.9)

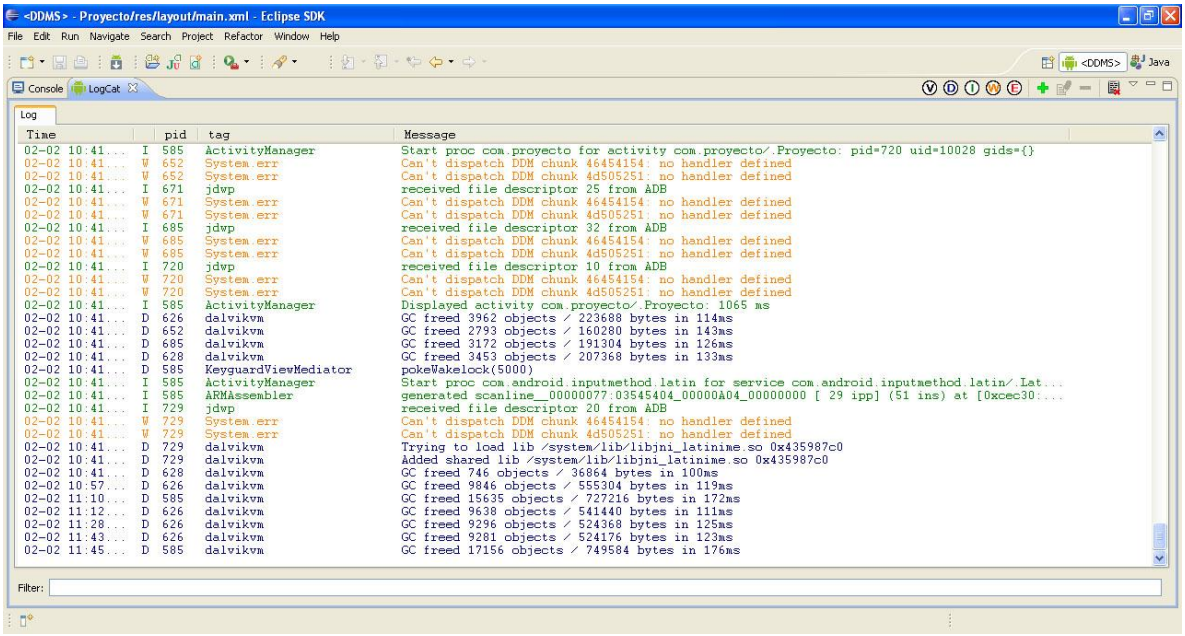


Figura 2.9 Dalvik Debug Monitor

Esta práctica conlleva numerosas ventajas a la hora de depurar ya que permite imprimir mensajes a lo largo de la ejecución. La manera de mostrarlos es mediante una clase del paquete *android.utils* llamada Log que es la encargada de facilitar el filtrado de los mensajes.

```
Log.v(tag, coords);
```

Código 2.1. Salida por pantalla de mensajes para la depuración de código.

Donde *tag* representa el nombre que se le desea dar a la etiqueta del Log y coords la variable a mostrar que podría ser una cadena de texto si fuese entre comillas.

El resultado mostrado por pantalla sería el siguiente:

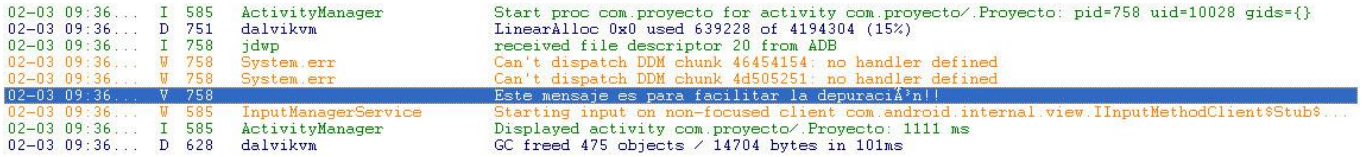


Figura 2.10 Impresión de mensajes en DMMS.

También es posible hacer la impresión con la sentencia de Java System.out.println("mensaje"); pero el inconveniente principal es a la hora de filtrar los mensajes ya que este método no dispone de etiquetas.

```
System.out.println("mensaje a mostrar");
```

Código 2.2. Salida por pantalla simple.

2.2 ECLIPSE

Figura 2.11 Logo de Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

El entorno fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Actualmente está siendo desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios tales como el caso que nos ocupa. El componente de Android para eclipse es una de las más potentes herramientas de programación en la actualidad. Instalando el SDK de Android en Ecplise este te permite crear y depurar aplicaciones con mayor eficiencia. Usando eclipse se tienen las siguientes ventajas:

-Permite el acceso a otras herramientas para desarrollar Android desde el entorno eclipse. Por ejemplo el ADT permite el acceso a muchas de las competencias de la herramienta DDMS.

- Proporciona ayuda a la hora de crear proyectos y crea automáticamente todos los archivos necesarios para un Nuevo proyecto de Android.

- Automatiza y simplifica el proceso de construir aplicaciones para Android.

- Provee con editores de XML para AndroidManifest.xml y otros archivos de recursos. También es capaz de exportar tu proyecto a un paquete que puede ser distribuido a otros usuarios.

2.3 Apache

Figura 2.12 Logo Apache.

Apache Web Server, es un servidor de páginas Web desarrollado por la Apache Sotfware Fundation, organización formada por miles de voluntarios que colaboran para la creación de software de libre distribución.

Es uno de los servidores más utilizados en Internet ya que se trata de un servidor muy potente, flexible, rápido, eficiente y que siempre está adaptado a nuevos protocolos http. Apache se encuentra disponible para varias plataformas, desde Debian, hasta Windows XP y se le puede incrustar nuevos módulos que le permitirán ejecutar código Script como son JSP, PHP, etc.

Entre las ventajas que presenta, destacan las siguientes:

- Modular

- Open source

- Multi-plataforma

- Extensible

- Gratuito

2.4 MySQL

Figura 2.13 Logo Mysql.

A continuación se realiza una descripción del sistema de gestión de bases de datos SQL de código abierto más popular.

MySQL es desarrollado, distribuido y soportado por MySQL AB, que es una compañía comercial Open Source de segunda generación que une los valores y metodología Open Source. Fue fundada por los desarrolladores de MySQL.

MySQL es un sistema de gestión de bases de datos:

Una base de datos es una colección estructurada de datos. Puede ser cualquier cosa, desde una simple lista de compra a una galería de pintura o las más vastas cantidades de información en una red corporativa. Para añadir, acceder, y procesar los datos almacenados en una base de datos se necesita un sistema de gestión de base de datos como MySQL Server. Al ser los computadores muy buenos en tratar grandes cantidades de datos, los sistemas de gestión de bases de datos juegan un papel central en computación, como aplicaciones autónomas o como parte de otras aplicaciones.

MySQL es un sistema de gestión de bases de datos relacionales:

Una base de datos relacional almacena datos en tablas separadas en lugar de poner todos los datos en un gran almacén. Esto añade velocidad y flexibilidad. La parte SQL de "MySQL"se refiere a "Structured Query Language". SQL es el lenguaje estandarizado más común para acceder a bases de datos y está definido por el estándar ANSI/ISO SQL. El estándar SQL ha evolucionado desde 1986 y existen varias versiones. En este manual, "SQL-92" se refiere al estándar del 1992, "SQL:1999" se refiere a la versión del 1999, y "SQL:2003" se refiere a la versión actual del estándar. Usamos la frase "el estándar SQL" para referirnos a la versión actual de SQL.

MySQL software es Open Source:

Open Source significa que es posible para cualquiera usar y modificar el software. Cualquiera puede bajar el software MySQL desde internet y usarlo sin pagar nada. Si lo desea, puede estudiar el código fuente y cambiarlo para adaptarlo a sus necesidades. El software MySQL usa la licencia GPL (GNU General Public License), http://www.fsf.org/licenses/, para definir lo que puede y no puede hacer con el software en diferentes situaciones. Si no se encuentra cómodo con la GPL o necesita añadir código MySQL en una aplicación comercial, puede comprarnos una licencia comercial. Consulte la Introducción a las Licencias MySQL para más información (http://www.mysql.com/company/legal/licensing/).

El servidor de base de datos MySQL es muy rápido, fiable y fácil de usar:

El servidor MySQL también tiene una serie de características prácticas desarrolladas en cooperación con los usuarios. Puede encontrar comparaciones de rendimiento de MySLQL Server con otros sistemas de gestión de bases de datos en nuestra página de comparativas de rendimiento.

MySQL Server se desarrolló originalmente para tratar grandes bases de datos mucho más rápido que soluciones existentes y ha sido usado con éxito en entornos de producción de alto rendimiento durante varios años. MySQL Server ofrece hoy en día una gran cantidad de funciones. Su conectividad, velocidad, y seguridad hacen de MySQL Server altamente apropiado para acceder bases de datos en Internet

MySQL Server trabaja en entornos cliente/servidor o incrustados:

El software de bases de datos MySQL es un sistema cliente/servidor que consiste en un servidor SQL multi-threaded que trabaja con diferentes bakends, programas y bibliotecas cliente, herramientas administrativas y un amplio abanico de interfaces de programación para aplicaciones (APIs).

También proporcionamos el MySQL Server como biblioteca incrustada multi-threaded que puede lincar en su aplicación para obtener un producto más pequeño, rápido y fácil de administrar.

Una gran cantidad de software de contribuciones está disponible para MySQL:

Lo que implica que la mayor parte de lenguajes de programacion y softwares que se utilizan muy a menudo tengan disponibilidad para enlazar con MySQL.

2.5 PHPMyAdmin

Figura 2.14 Logo phpMyAdmin.

Es la herramienta Web que se ha usado para controlar y manejar las bases de datos MySQL. Como la base de datos ha sido creada a través de CakePhp, esta herramienta, aunque también se podría haber usado para realizar las tareas de creación, ha sido mas usada para modificar los campos de eventos y crear rutas, es decir para poder testear y simular la aplicación móvil. No la aplicación móvil en si, sino los controladores que son los que realmente se comunican con la base de datos.

PhpMyAdmin es un programa de libre distribución en PHP, es una herramienta muy completa que permite acceder a todas las funciones típicas de la base de datos MySQL a través de una interfaz Web muy intuitiva.

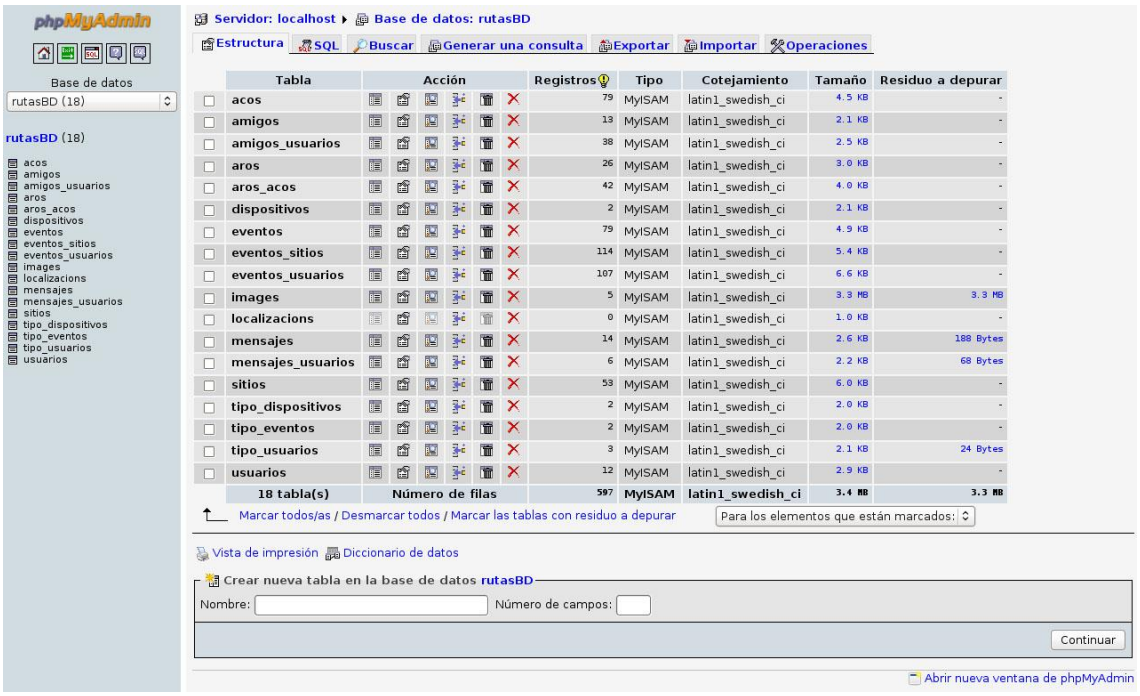


Figura 2.15 Interfaz gráfica de phpMyAdmin.

Esta herramienta permite crear y eliminar bases de datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos y está disponible en 50 idiomas.

2.5 ZXING

ZXing es una librería Opensource hecha en Java capaz de procesar códigos de barra en 1D y 2D. La librería está orientada en proporcionar soluciones de escaneo para terminales móviles, es decir, codifica y decodifica códigos de barras.

Los formatos que soporta la librería son los siguientes:

- UPC-A and UPC-E
- EAN-8 and EAN-13
- Code 39
- Code 93
- Code 128
- ITF
- Codabar
- RSS-14 (all variants)
- RSS Expanded (most variants)
- QR Code
- Data Matrix
- Aztec ('beta' quality)
- PDF 417 ('alpha' quality)

La libreria estadividida entre varios components principals, son los siguientes:

- **core**: La libreria de decodificacion de imagen y testeo de código
- **javase**: J2SE- código de cliente especifico
- **zxingorg**: El source de `zxing.org/w`
- **zxing.appspot.com**: El código de la apliación web de generación de códigos
- **android**: El cliente de android, conicido como Barcode Scanner
- **androidtest**: Android test app
- **android-integration**: Ofrece la integracion del barcode Scanner mediante intents

CAPÍTULO 3: DESARROLLO

3.1 Implementación

El funcionamiento general de la aplicación es el siguiente:

- El usuario enciende su ordenador y abre su navegador Web.

- Se dirige a la página donde está alojada la aplicación (http://pcacribia.upct.es/rutas2/) y se conecta con el servidor.

- El usuario se inicia sesión en su cuenta, o se registra con una nueva. Al hacer esto, el servidor está mandando los datos del usuario a la Base de Datos.

- La Base de Datos ejecuta la consulta y devuelve la información al servidor.

- En el caso de que el usuario se haya conectado correctamente con sus datos, la información devuelta es la redirección a su página de perfil, mostrándolo al usuario en la pantalla de su ordenador.

- En un momento dado, el usuario va caminando por la calle e inicia la aplicación móvil.

- El usuario elige cualquier opción del menú de la aplicación móvil. En ese momento, el móvil se conecta al servidor, éste a su vez se conecta a la Base de Datos, ejecuta la consulta solicitada y devuelve la información obtenida.

- Si el usuario ha seleccionado la opción "Comprobar puntos cercanos", la aplicación se conecta con el GPS y compara la información devuelta por la Base de Datos con la información que le proporciona el GPS.

- Finalmente, la información es mostrada al usuario a través de la pantalla del móvil.

El desarrollo de la aplicación móvil, responde al siguiente diagrama de flujo, gracias al cual se comprenderán mejor los siguientes apartados.

4:11 PM
Login
Usuario
admin@hotmail.com
Contraseña
Enviar
4:14 PM
Login
Mis eventos
Mis eventos no iniciados
Mis eventos en curso
Mensajes recibidos
Mensajes enviados
Comprobar puntos cercanos
Configuracion de ubicacion
6:40 PM
Proyecto
Error usuario o contraseña
6:40 PM
Proyecto
Se ha completado la ruta
6:40 PM
Proyecto
El código promocional es: X002BD34
4:14 PM
Security & location
My Location sources
Use wireless networks
Use WLAN, Mobile networks to help Maps using wireless networks
Enable GPS satellites
When location accurate to street level (deselect to conserve battery)
Screen unlock pattern
Require pattern
Must draw pattern to unlock screen
Use visible pattern
Use tactile feedback
6:23 PM
Mensaje 1
Mensaje 2
Mensaje 3
Mensaje 4
Mensaje 5
Mensaje 6
Mensaje 7
Mensaje 8
Mensaje 9
Mensaje 10
6:09 PM
Wireless & networks
Airplane mode
Wi-Fi
Wi-Fi settings
Bluetooth
Bluetooth settings
VPN settings
Mobile networks
6:21 PM
Evento numero 1
Evento numero 2
Evento numero 3
Evento numero 4
Evento numero 5
Evento numero 6
Evento numero 7
Evento numero 8
Evento numero 9
Evento numero 10
6:21 PM
Evento numero 1
Evento numero 2
Evento numero 3
Evento numero 4
Evento numero 5
Evento numero 6
Evento numero 7
Evento numero 8
Evento numero 9
Evento numero 10
6:21 PM
Evento numero 1
Evento numero 2
Evento numero 3
Evento numero 4
Evento numero 5
Evento numero 6
Evento numero 7
Evento numero 8
Evento numero 9
Evento numero 10
Canada
United States
Brasil

3.1.1 Interfaz de usuario

Para la realización del proyecto se han definido vistas y diseños en XML en lugar de hacerlo en Java. El motivo de hacerlo de esta manera es que resulta más sencillo de definir además de que sirve para desvincular código, y sobre todo que se puede reutilizar en otros contextos. Los archivos de recursos de vista se incluyen en el directorio res/layout. La raíz de estos archivos XML suele ser una de las subclases de diseño ViewGroup tipo RelativeLayout, LinearLayout, FrameLayout. Conviene mencionar que no es necesario que los recursos del directorio res/layout sean diseños, por ejemplo, puede definir un TextView.

La manera de definir dentro del archivo la interfaz se define mediante tags. Es posible realizar el diseño de la interfaz grafica mediante el Plug-In ADT para Eclipse ofrece un editor gráfico con el que se puede componer pantallas fácilmente, siendo por otra parte conveniente saber editar con código.

Por la parte de código, un ejemplo de archivo es el que se genera automáticamente al crear un nuevo proyecto nuevo:

```
<?xml version=”1.0″ encoding=”utf-8″?>
<LinearLayout xmlns:android=”http://schemas.android.com/apk/res/android”
android:orientation=”vertical”
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”>

<TextView
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”@string/hello”/>
</LinearLayout>
```

Código 3.1. Archivo main.xml autogenerado.

Para asociar un archivo de layout con una actividad solo hay que escribir lo siguiente en el código de la Actividad:

```
public void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.main);
}
```

Código 3.2. Código autogenerado en la clase fuente.

La manera de hacer referencia a estos archivos de diseño es mediante el tipo y el ID. Para nuestro proyecto el ViewGroup más utilizado ha sido Linear Layout. También se definen LayoutParams en XML con la convención android:layout_[atributo] , donde [atributo] hace referencia a un atributo de diseño. Junto con el diseño se pueden definir otros atributos relacionados con View en XML por medio de atributos equivalentes a los métodos disponibles en código, por ejemplo Android: padding sería setPadding().

Tras definir el LinearLayout principal se añaden los elementos View secundarios. En este caso utilizamos TextView cada uno con su propio Id android:id="@+id/[nombre]". Al establecer un Id de esta forma se define una referencia Int en la tabla de recursos y se le asigna el nombre especificado. De este modo otros componentes pueden hacer referencia al Id por medio del nombre textual. Una vez definidas las vistas como recursos se puede utilizar el método findViewByID() de Activity para obtener una referencia a una vista concreta por medio del nombre pudiendo modificarla después en el código.

```
this.user = (EditText) findViewById(R.id.get_input_user);
```

Código 3.3. Relación findViewById().

De este modo se amplia y entrega el elemento get_input_user. Es interesante aclarar, que las vistas secundarias de los archivos de diseño terminan como tipo id en R.java es decir, no son R.layout.name sino R.id.name, aunque estén en el directorio res/layout.

En el archivo XML definiremos el EditText de la siguiente manera:

```
<EditText android:id="@+id/get_input_user" android:layout_width="fill_parent"
android:layout_height="wrap_content" android:layout_marginBottom="10px" />
```

Código 3.4. Ejemplo del elemento EditText.

Algo a destacar de este fragmento de código de los archivos XML de la carpeta res/layout es la sintaxis @id. Esta referencia apunta a otros elementos de recurso desde el archivo de recursos actual.

Un ejemplo con elementos variados seria el siguiente que vamos a mostrar con la particularidad de poseer un TextView, un EditText y un Boton:

```
<?xml version=”1.0″ encoding=”utf-8″?>
<LinearLayout
xmlns:android=”http://schemas.android.com/apk/res/android”
android:orientation=”vertical”
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
android:padding=”10px”>


<TextView
android:text=”@string/introduccion”
android:id=”@+id/TextViewIntroduccion”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”></TextView>


<EditText
android:text=”@string/prueba”
android:id=”@+id/editTextPrueba”
android:layout_height=”wrap_content”
android:layout_width=”fill_parent”
android:layout_marginTop=”5px”></EditText>


<Button
android:text=”@string/boton”
android:id=”@+id/buttonPrueba”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
android:clickable=”true”
android:layout_gravity=”right”
android:layout_marginTop=”5px”></Button>
</LinearLayout>
```

Código 3.5. Ejemplo avanzado con TextView, editText y Button.

Figura 3.1 Representación gráfica de los elementos TextView, EditText y Button.

Referencias estáticas

Algunas de las vistas representan etiquetas que se muestran en la pantalla tal cual y no se manipulan en el código. Estos elementos con valores conocidos, se definen con referencias a cadenas externas, es decir, están definidas fuera del código. La definición de interfaces de Android permite separar los elementos gráficos del código, la manera de realizar esta acción es introducir las cadenas necesarias en el archivo res/values/strings.xml:

```
<?xml version=”1.0″ encoding=”utf-8″?>
<resources>
<string name=”app_name”> Aplicación </string>
<string name=”introduccion”>Introduce lo que desees y pulsa el botón</string>
<string name=”boton”>Boton</string>
<string name=”prueba”> prueba</string>
</resources>
```

Código 3.6. Ejemplo de referencia estática.

3.1.2 Gestión de eventos

Los eventos sirven para mejorar la interacción del usuario con el terminal. Indican, por ejemplo, que alguien ha pulsado un botón. Èstos se gestionan a través de Listeners. Las clases que lo utilicen deberán implementar los listeners necesarios (uno por cada uno de los eventos posibles), además de implementar los métodos que capturarán los eventos.

Los eventos se dividen en dos partes: el componente que genera el evento y el componente (o componentes) que responde al mismo. Estas dos partes se denominan Observable y Observer en términos de diseño.

Un componente observable permite que las instancias Observer se registren. Al producirse un evento, Observable notifica a todos los observadores que ha pasado algo. Tras ello, los observadores responden a dicha notificación como consideran oportuno. Con respecto a un botón de Android, las dos partes se representan de esta forma:

-Observable: Button.setOnclickListener(OnclickListener listener)
-Observer: listener: listener.onclick(View v)

Este patrón aparece en los elementos View de Android ya que muchos elementos se consideran Observable y permiten que otros componentes se puedan conectar y escuchar eventos. Por ejemplo, la mayoría de los métodos de la clase View que empiezan por "on" están relacionados con evento: onFocusChanged(), onSizeChanged(), onLayout(), onTouchEvent() y similares.

Incluidos en los listeners de eventos están los siguientes métodos:

- **onClick () de View.OnClickListener**. Es llamado cuando el usuario toca el item, o se centra en el elemento con las teclas o rueda de desplazamiento y presiona la tecla “enter” o presiona la rueda de desplazamiento.

- **onLongClick() de View.OnLongClickListener**. Esta llamada ocurre cuando el usuario pincha sobre un item un largo periodo de tiempo (un segundo). Ya sea con la tecla “enter” o presionando la rueda de desplazamiento

-**onFocusChange() de View.OnFocusChangeListener**. Esta llamada ocurre cuado el usuario navega haciendo zoom o quitando zoom sobre un item.

-**onKey() de View.OnKeyListener**. Esta llamada ocurre cuando el usuario ha seleccionado un item y presiona un botón del dispositivo.

-**onTouch() de View.OnTouchListener**. Esta llamada ocurre cuando el usuario realiza una acción que conlleve algún movimiento como la presión o tocado de pantalla, incluso cualquier tipo de movimiento del móvil.

-**onCreateContextMenu() From View.OnCreateContextMenuListener**. This is called when a Context Menu is being built (as the result of a sustained "long click"). See the discussion on context menus in Creating Menus for more information.

-**onCreateContextMenu() de View.OnCreateContextMenuListener**. Esta acción es llamada cuando un Context Menu esta siendo construido.

Igualmente, los métodos de ciclo de vida de Activity como onCreate() y onFreeze() están relacionados con eventos, a otro nivel.

Los eventos no solo se producen en la interfaz de usuario sino en toda la plataforma. Por ejemplo, cuando se recibe una llamada o una ubicación GPS cambia por un movimiento físico, se producen diversas reacciones; muchos componentes pueden ser notificados al sonar el teléfono o cuando cambia la ubicación (no solo uno y no solo la UI).

Las vistas admiten eventos en distintos niveles. Cuando se produce un evento de interfaz (el usuario pulsa un botón, se desplaza o selecciona una parte de una ventana), se comunica a la vista correspondiente. Por lo general, los eventos mas comunes (y los que han sido usados en el proyecto) son los eventos de clic, de teclado y táctiles.

Los pasos para, por ejemplo, capturar un botón, serían los siguientes:

1. Hacer que la clase implemente los listeners necesarios y añadir los métodos que se requieran (Eclipse puede hacerlo automáticamente).
2. Capturar los controles de la interfaz en el método onCreate, y asociar sus listeners a la clase.
3. Implementar los gestores de eventos para cada control en cada uno de los métodos de los listeners.

```
public class ExampleActivity extends Activity implements OnClickListener {
  protected void onCreate(Bundle savedValues) {
     ...
     Button boton = (Button)findViewById(R.id.boton);
     boton.setOnClickListener(this);
  }

  // Implement the OnClickListener callback
boton.setOnItemClickListener(new OnItemClickListener() {
          public void onItemClick(AdapterView<?> parent, View view,
              int position, long id) {
```

```
            // Se muestra un mensaje al clicar el boton
            Toast.makeText(getApplicationContext(), ((TextView)
view).getText(),
                  Toast.LENGTH_SHORT).show();
          }
        });


 ...
}
```

Código 3.7. Ejemplo de captura de evento con un botón.

3.1.3 List View

Cada uno de los submenús forman parte de una ListView con el unico fin de facilitar la labor de descargar la información y representarla en las lista. Gracias a un ListAdapter los elementos de una cadena de texto se insertan automáticamente en la lista.

Al igual que con los mapas, que extienden de de MapActivity, para mostrar un ListView es necesario extender la clase de ListActivity en vez de Activity.

Esta aplicación consta de dos tipos de listas, estáticas y dinámicas, para el menú principal se ha implementado una lista estática. Para ello se ha introducido una cadena de texto con cada uno de los elementos del menú en la variable MENU, una cadena de texto cuyo atributo es estático para que no sea posible su modificación a nivel de código. El resto de listas son dinámicas ya que su función es representar los elementos descargados del servidor, teniendo, de este modo la propiedad de ser dinámicos. El hecho de tener esta característica es esencial ya que, por ejemplo, un usuario puede estar inscrito a quince eventos y otro a dos solamente, la lista se adapta al número de eventos.

Para crear una lista, lo primero que deberemos hacer es crear un layout en nuestra carpeta res/layout del proyecto insertando el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
```

```
    </TextView>
```

Código 3.8 Archivo xml para la creación de una lista.

SetListAdapter (ListAdapter) agrega automáticamente un ListView para llenar toda la pantalla de la ListActivity. Este método tiene una ArrayAdapter, que gestiona el conjunto de elementos de la lista, que se pondrán en el ListView. El constructor ArrayAdapter toma el contexto de aplicación, la descripción del diseño de cada elemento de la lista, y una lista de objetos para insertar en el ListView.

3.1.4 Conexiones HTTP

Para el correcto funcionamiento de la aplicación es necesario, en la mayoría de los casos y ya que la aplicación no dispone de base de datos, descargar los datos del servidor. El método de intercambio de información se realiza mediante el protocolo HyperText Transfer Protocol o HTTP (en español protocolo de transferencia de hipertexto). Este protocolo esta orientado a realizar transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. En este el móvil es el cliente, quien efectúa las peticiones. Estas peticiones se efectúan a una URL o localizador uniforme de recursos, es este caso se trata del servidor ubicado en la UPCT con URL http://pcacribia.upct.es. Luego de interpretar la petición el servidor genera una respuesta, enviándola de nuevo al terminal móvil.

En este caso el establecimiento de la conexión para los intercambios de información se realizan en texto plano mediante el método GET. Para ello usaremos la clase HttpConnection.

Para hacer uso de estas clases se debe importar al programa las clases específicas para la conexión, como se muestra en el código 3.9:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
```

Código 3.9. Imports de la clase donde se va a implementar la conexión.

Para poder realizar una conexión es necesario haber declarado en el manifiesto el permiso que da acceso a Internet android.permission.INTERNET. Este permiso ya fue declarado con anterioridad para poder hacer uso de los servicios de Google Maps.

```
private InputStream OpenHttpConnection(String urlString)
          throws IOException
          {
              InputStream in = null;
              int response = -1;

              URL url = new URL(urlString);
              URLConnection conn = url.openConnection();

              if (!(conn instanceof HttpURLConnection))
                  throw new IOException("No se puede conectar");

              try{
                  HttpURLConnection httpConn = (HttpURLConnection) conn;
                  httpConn.setAllowUserInteraction(false);
                  httpConn.setInstanceFollowRedirects(true);
                  httpConn.setRequestMethod("GET");
                  httpConn.connect();

                  response = httpConn.getResponseCode();
                  if (response == HttpURLConnection.HTTP_OK) {
                      in = httpConn.getInputStream();
                  }
              }
              catch (Exception ex)
              {
                  throw new IOException("Error connecting");
              }
              return in;
          }

          private String DownloadText(String URL)
          {
              int BUFFER_SIZE = 2000;
              InputStream in = null;
              try {
                  in = OpenHttpConnection(URL);
              } catch (IOException e1) {
                  e1.printStackTrace();
                  return "";
              }

              InputStreamReader isr = new InputStreamReader(in);
              int charRead;
                String str = "";
                char[] inputBuffer = new char[BUFFER_SIZE];
              try {
                  while ((charRead = isr.read(inputBuffer))>0)
                  {
                      //---convert the chars to a String---
                      String readString =
```

```
                        String.copyValueOf(inputBuffer, 0, charRead);
                    str += readString;
                    inputBuffer = new char[BUFFER_SIZE];
                }
                  in.close();
            } catch (IOException e) {
                e.printStackTrace();
                return "";
            }

            return str;
        }
```

Código 3.10. Conexión HTTP para recibir los datos.

En el caso de no ser posible la conexión por circunstancias varias saltaría la excepción que debe ser capturada para que no se produzca una interrupción en la ejecución del programa. Esta excepción redirige a la pantalla de configuraciones inalámbricas para, de este modo, activarlas y poder proseguir con la ejecución del programa.

```
    try{

        //código pertinente

        }catch(NullPointerException npe){


                startActivityForResult(new
Intent(android.provider.Settings.ACTION_WIRELESS_SETTINGS), 0);


            }
```

Código 3.11. Captura de una NullPointerException.

3.1.5 GPS

La finalidad de utilizar la señal GPS en la aplicación es para conocer la ubicación del usuario, sabiendo esto, es posible hacer una búsqueda en el servidor de los eventos cercanos a los que esta apuntado y así alertar de su proximidad.

El paquete android.location es el encargado de proporcionar las clases para poder acceder y controlar la señal del GPS. Cabe destacar que este paquete no solo aporta clases para el GPS, sino además, para otro tipo de dispositivos de localización.

Este paquete contiene varios elementos en los que vamos a profundizar a continuación:

- **Location**: representa localizaciones geográficas.

- **LocationManager**: gestiona los dispositivos de localización.

- **Geocoder**: traduce las direcciones físicas y coordenadas en latitud y longitud, y viceversa.

- **LocationListener**: interfaz que permite implementar una clase para la captura de eventos asociados al dispositivo de localización.

Para obtener la señal del dispositivo y todo lo referente a su gestión se usa la clase LocationManager, y la clase LocationListener para hacer eco de los cambios que se quieran capturar, como por ejemplo un cambio de posición. Mientras que la clase Location es la muestra toda la información en los mapas.

Debido a que la librería de mapas no es una parte de la librería estándar de Android hay que declararlo en el Andriod Manifest. Añadiendo esta línea como hijo del nodo <aplication>.

```
<uses-library android:name="com.google.android.maps" />
```

Código 3.12. Declaración en el AndroidManifiest.xml para el acceso a los mapas.

Todas estas clases están implementadas en el proyecto en una clase llamada GPS.java desde donde se controla el GPS del dispositivo gracias a la variable lm.

Se considera importante explicar detenidamente todo el funcionamiento de locationManager ya que es la parte mas importante para acceder a los servicios de localización y el estado del GPS. El método getSystemService() de la clase Activity es el encargado de acceder al dispositivo de localización. además tendremos que pasarle como

argumento la constante context.LOCATION SERVICE para referenciar al GPS. Plasmado en código fuente todo lo explicado quedaría plasmado en esta línea:

```
lm = (LocationManager) getSystemService(LOCATION_SERVICE);
```

Código 3.13. Declaración en el manifiesto del permiso de acceso al GPS

Una vez creado un controlador válido para el dispositivo GPS hay que definir las clases para la gestión de posibles eventos asociados al GPS tales como un cambio de posición (cuando el usuario se desplaza) o simplemente la pérdida de cobertura del GPS provocando la desactivación del mismo.

Para ir actualizando la posición del GPS en nuestra clase GPS.java se ha definido el metodo onResume() el cual es llamado después de onStart incluso si la aplicación no ha sido detenida. RequestLocationUpdates se encarga de actualizar la posición cada 1000ms o conun desplazamiento de superior a diez metros

```
protected void onResume() {

      lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000,
10f, this);

                  super.onResume();
            }
```

Código 3.14 Método requestLoctionUpdates().

Una vez declarados todos estos métodos y variables ya es posible acceder a la ubicación del usuario gracias al método getLastKnownLocation() asociado al objeto lm que es el que controla el dispositivo.

El objetivo de conocer la posición del usuario es para compararla con la posición de los eventos cercanos. Por lo que se ha creado el método proximityAlert ().

La forma de obtener la posición viene dada por el objeto location al cual se le asignan los métodos getLatitude() y getLongitude(). Así mismo las latitudes y las longitudes de los distintos eventos se comparan con la localizacion actual del dispositivo.

Si alguno de estos eventos esta dentro del radio (definido con la variable radius de tipo int) que en este caso se le ha atribuido un valor de 500 metros. La forma de discernir si la distancia entre ambos puntos es mayor o menor del valor indicado es mediante el Teorema de Pitágoras

Sean dos puntos $A(x_1, y_1)$ y $B(x_2, y_2)$

La distancia vendrá dada la siguiente expresión:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

En caso de ser la distancia mayor que 500 metros el método devolverá un 0 por lo que no saltará ninguna alarma, de no ser así, es decir, una distancia menor, la aplicación saltará a la clase Alerta.java donde se mostrará lo necesario para que el usuario sepa que ese evento ha saltado. Ver código 3.15

```
public int proximityAlert(double latitude, double longitude, double
radius){

                  double aux;
                  double distancia;
                  Location location =
lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);     double
double ActualLatitude = location.getLatitude()/ 1000000 ;
double ActualLongitude = location.getLongitude()/ 1000000 ;

aux = (ActualLatitude - latitude) * (ActualLatitude - latitude) +
(ActualLongitude - longitude) * (ActualLongitude - longitude);
      distancia = Math.sqrt(aux);

                  if (distancia <= radius){
                        return 1;
                  }else{
                  // NO SALTA LA ALERTA
                        return 0;
                  }
            }
```

Código 3.15 Implementación del método proximityAlert().

Es necesario incluir el método onLocationChanged aunque en este caso como el usuario va a consultar los eventos cercanos estando en un mismo sitio no hace falta implementar nada.

A continuación se muestra el código referente:

```
@Override
   public void onLocationChanged(Location location) {
            Log.v(tag, "Ha habido un cambio de localizacion");
```

```
        }
```

Código 3.16 Implementación del método onLocationChanged.

Es necesario incluir un método para detener el funcionamiento del GPS ya que uno de los problemas vinculados a su uso es el gran gasto de batería que supone su funcionamiento. La manera de apagarlo es dejando de recibir actualizaciones de posición y haciendo una llamada a onResume() tal que así:

```
protected void onPause() {

                lm.removeUpdates(this);
                super.onResume();
        }
```

Código 3.17 Implementación del código onPause().

Además se ha incluido código para llamar a la configuración del GPS cuando este está deshabilitado para impedir bloqueos en el programa.

```
public void onProviderDisabled(String provider) {


      Intent intent = new Intent(

      android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                startActivity(intent);
        }
```

Código 3.18 Implementación del método onProviderDisabled.

3.1.6 Estado del GPS

Al estar en continuo movimiento, el GPS puede sufrir cambios de estado debido a la variabilidad de cobertura pasando, por ejemplo, de estar activo a inactivo en pocos metros. El método onStatusChanged se mantiene a la escucha y cuando el GPS se desactiva éste capta el cambio y muestra las alertas pertinentes. En concreto este método es capaz de captar tres tipos de cambios:

- `OUT_OF_SERVICE`: Alarma que salta cuando el GPS está fuera de servicio.

- `TEMPORARILY_UNAVAILABLE`: Cuando el GPS no esta disponible.

- AVAILABLE: Alarma para cuando el GPS esta disponible.

A continuación se muestra el código que implementa lo anteriormente descrito:

```
public void onStatusChanged(String provider, int status, Bundle
extras) {
                  /* This is called when the GPS status alters */
      switch (status) {
            case LocationProvider.OUT_OF_SERVICE:
                  Log.v(tag, "Status Changed: Out of Service");
                  Toast.makeText(this, "Status Changed: Out of
Service",
                  Toast.LENGTH_SHORT).show();
                  break;

            case LocationProvider.TEMPORARILY_UNAVAILABLE:
            Log.v(tag, "Status Changed: Temporarily Unavailable");
                  Toast.makeText(this, "Status Changed: Temporarily
Unavailable",
                  Toast.LENGTH_SHORT).show();
                  break;

            case LocationProvider.AVAILABLE:
                  Log.v(tag, "Status Changed: Available");
                  Toast.makeText(this, "Status Changed: Available",
                  Toast.LENGTH_SHORT).show();
                        break;
                  }
            }
```

Código 3.19 Comprobación del estado del GPS

3.1.7 Google Maps

Una de las características más importantes de esta aplicación es la utilización de los mapas para representar la posición exacta de los eventos.

Como se indica en el primer capitulo, apartado de motivación, una de las características principales por las que se escogió Android entre otras es esta misma. Los mapas. El paquete que Google nos proporciona para controlar y cargar los mapas es com.google.android.maps.

Dentro de ese paquete se encuentran todas las clases necesarias para controlar el mapa de entre las cuales podemos nombrar algunas:

- MapView: obtiene el mapa solicitado y lo muestra en pantalla.

- `MapController`: gestiona los mapas, como desplazamientos o zoom.

- `GeoPoint`: sirve para representar un punto en el mapa según su latitud y longitud.

- `Overlay`: nos ayuda a representar elementos gráficos sobre el mapa.

- `MapActivity`: clase que extiende la clase base Activity, y permite crear una Activity con un mapa.

Para poder usar todas estas clases en el emulador es necesario registrarse en Google maps para obtener la llamada API Key. Sin la API KEy no podremos trabajar con los mapas, siendo esta única para todo el proyecto.

Para obtenerla, es necesario tener una cuenta en Google y aceptar las normas de uso. La finalidad de esta llave es garantizar el uso adecuado de los servicios y la información que Google nos va a facilitar además de dotar al programador la autoría del proyecto.

Para la obtención de la huella digital ha sido preciso escribir lo siguiente en el terminal de Linux:

```
cd /home/rutas/.android
$ keytool -list -keystore debug.keystore
```

Código 3.20 Keytool para generar la clave

Una vez obtenido el certificado digital se introduce en el formulario de obtención de la API en la dirección **http://code.google.com/intl/es-ES/android/maps-api-signup.html** y obtenemos el siguiente resultado:

```
Gracias por suscribirte a la clave del API de Android Maps.

Tu clave es:

0JosgJ6dQOtHkWBLVh1D3z81uwWndiX8dlHkXaw

Esta clave es válida para todas las aplicaciones firmadas con el certificado cuya huella dactilar sea:

93:FA:74:AD:09:5E:25:15:AC:E2:67:C6:9D:6A:6E:35
```

Incluimos un diseño xml de ejemplo para que puedas iniciarte por los senderos de la creación de mapas:

```
<com.google.android.maps.MapView
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:apiKey="0JosgJ6dQOtHkWBLVh1D3z81uwWndiX8dlHkXaw"
  />
```

Consulta la documentación del API para obtener más información.

Código 3.21 Ejemplo de *API key* proporcionada por Google.

Mostrar un mapa

Figura 3.2. Ejemplo de MapView.

Una vez se dispone de una API Key se puede comenzar a programar el acceso a los mapas de Google Maps. Usando el paquete com.google.android.maps. Se describe a

continuación con una muestra de los paquetes importados en el código de la clase que implementa el MapView para su correcto funcionamiento

```
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.OverlayItem;
```

Código 3.22 Clases necesarias para mostrar un mapa de Google Maps.

La principal característica de las clases que muestran mapas es que a diferencia de las demás de la aplicación que extienden de Activity, estas extienden de MapActivity que es la relacionada con los mapas. Vamos a continuación a hablar de MapView

3.1.8 MapView

Es una versión en miniatura de los conceptos del API de Google Maps. Muestra sectores de un mapa que obtiene de la red al mover y ampliar el mapa, similar a la versión Web, aunque no idéntico.

Para poder hacer uso de todas estas librerías referentes a mapas necesitamos incluir el API Key obtenido en el archivo XML donde se define el layout de la siguiente manera:

```
<com.google.android.maps.MapView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:enabled="true"
android:clickable="true"
android:apiKey="example_Maps_ApiKey_String"
/>
```

Código 3.23 Implementación del API key.

Otra clase muy utilizada es GeoPoint, incluida en el paquete *com.google.android.maps*. La particularidad de esta clase es que representa un punto a través de sus coordenadas. Estas coordenadas vienen dadas por una latitud y longitud en un formato distinto al convencional, para su conversión al formato estándar es necesario multiplicarlas por la potencia 10^6.

Hemos dicho hasta ahora que MapActivity es lo que relaciona la aplicación con los paquetes de APIs que tienen las utilidades relacionadas con los mapas. MapActivity se encarga de forma transparente para nosotros de implementar muchas de las utilidades de un Mapview, que, como su propio nombre indica es la vista del mapa. MapActivity configura automáticamente los subprocesos de red y todas las tareas relacionadas con mapas.

Para poder usar este tipo específico de actividad es necesario habilitar el paquete com.google.android.maps en el manifiesto de la siguiente manera:

```
  <uses-library android:name="com.google.android.maps" />
```

Código 3.24 Declaración en el manifiesto del API de Google Maps

Una última cosa para mostrar el mapa es conceder permisos en el AndroidManifest.xml de nuestra aplicación tanto de acceso a mapas como de acceso a Internet:

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

Código 3.25. Declaración en el manifiesto del permiso de conexión a Internet

A continuación se muestra el código para mostrar un mapa, comentar que no está completo ya que esta demostración persigue exponer únicamente lo necesario para la implementación del mapa:

```
public class HelloMapView extends MapActivity {

      LinearLayout linearLayout;
      MapView mapView;
      List<Overlay> mapOverlays;
      Drawable drawable;
      HelloItemizedOverlay itemizedOverlay;


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

        mapView = (MapView) findViewById(R.id.mapview);
        mapView.setBuiltInZoomControls(true);

        mapOverlays = mapView.getOverlays();
        drawable =
this.getResources().getDrawable(R.drawable.androidmarker);
        itemizedOverlay = new HelloItemizedOverlay(drawable);
```

```
        Intent Dato= getIntent();
        String pos = Dato.getExtras().getString("posicion");
        int evento_id = Dato.getExtras().getInt("pos_evento");

        for(i= 0; i<m.length ;i++){


        GeoPoint point = new GeoPoint(lati,longi);
        OverlayItem overlayitem = new OverlayItem(point, "", "");

        itemizedOverlay.addOverlay(overlayitem);


        }
        mapOverlays.add(itemizedOverlay);

    }

    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
```

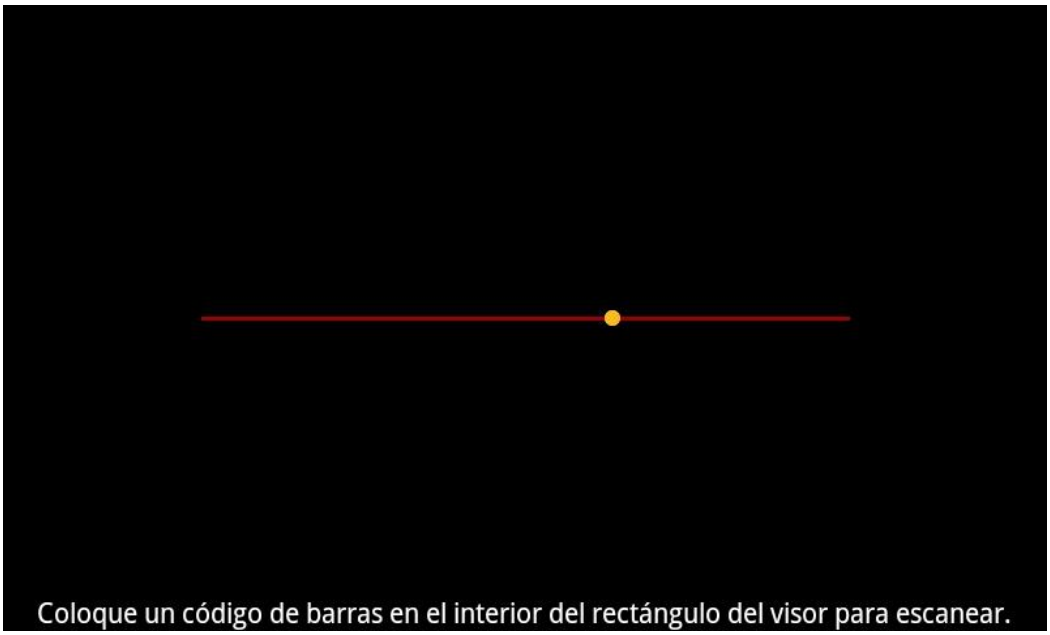
Código 3.26 Código necesario para mostrar un mapa de Google Maps

3.1.8 Uso de intents, integración con ZXING

La manera de integrar zxing con un intent es mediante el IntentIntegrator, es la mejor forma de integración usando librerías de código especialmente diseñadas para este tipo de casos. Son capaces de controlar muchas características como categorías, flags, capturar la aplicación adecuada y manejar correctamente el caso de que no esté instalada la aplicación de Barcode Scanner

Con el código siguiente se invoca el intent:

```
IntentIntegrator integrator = new
IntentIntegrator(yourActivity);
integrator.initiateScan();
```



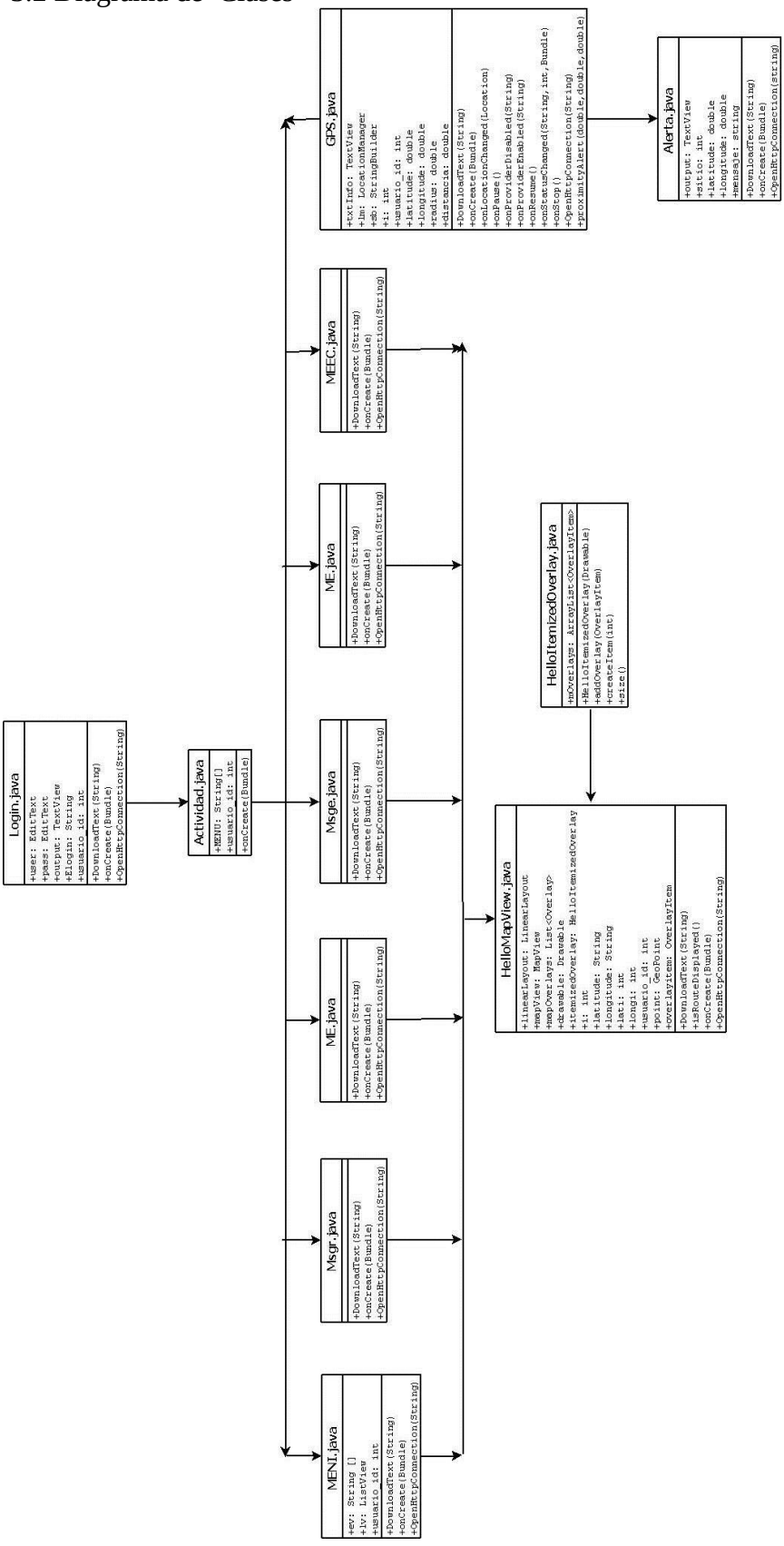
Una vez detectado el código se procederá a su decodificación y a mostrar el mensaje del resultado obtenido como se muestra a continuación.

Código para recoger los resultados de la captura de código de barras:

```
public void onActivityResult(int requestCode, int resultCode,
Intent intent) {
  IntentResult scanResult =
IntentIntegrator.parseActivityResult(requestCode, resultCode,
intent);
  if (scanResult != null) {
    // handle scan result
  }
  // else continue with any other code you need in the method
  ...
}
```

Este ejemplo es el básico, existen más métodos y opciones en esta clase que permiten configurar la respuesta a la lectura del código de barras o realizar otras acciones como compartir texto vía QR Codes.

3.2 Diagrama de Clases



3.3 Descripción de los Controladores

Este apartado persiguie describir la funcionalidad que ofrece cada controlador, asimismo se muestra un diagrama de relación entre las clases y los controladores.

- **Login**: recibe el email y la contraseña comparándolos en la base de datos, si ambos concuerdan imprime el numero del id de usuario, de lo contrario imprime un "0"

- **Mensajes_Enviados** Busca en la tabla mensajes usuarios todos los mensajes cuyo origen_id sea el mismo id del usuario recogiendo a su vez el id del mensaje en cuestión y buscando su descripción en la tabla mensajes.

- **Mensajes_Recibidos** Busca en la tabla mensajes usuarios todos los mensajes cuyo destino_id sea el mismo id del usuario recogiendo a su vez el id del mensaje en cuestión y buscando su descripción en la tabla mensajes.

- **Mis eventos completados** equivale a la clase mis eventos. Este controlador busca en la tabla eventos_usuarios todos los eventos relacionados con el usuario que haya iniciado sesión seleccionando solo los eventos cuyo valor del campo completado sea 1. Almacena el id de estos eventos con campo completado 1 y hace una búsqueda en la tabla eventos, mostrando así su descripción

- **Mis eventos en curso**: Busca en la tabla eventos_usuarios todos los eventos cuyo campo completado este a "0" y almacena su evento_id para luego hacer una búsqueda en la tabla eventos de donde recoge la variable Desc y la muestra por pantalla.

- **Mis eventos no iniciados** busca en eventos_usuario de los eventos del usuario conectado filtrándolo por los eventos cuyo campo eventos_sitio_id esté a "0" mientras va almacenando el campo evento_id para mostrar su descripción por pantalla.

- **Puntos de evento,** recibe de la clase el evento_id para el que ha de buscar todos los puntos que lo componen, para ello realiza una búsqueda en la tabla eventos_sitios almacenando todos los puntos relacionados con los sitios a los que pertenece el evento.

- **Coordenadas pendientes** este controlador recibe como parámetro el id del usuario y se dedica a buscar todos los eventos a los que se ha apuntado el usuario y no han sido completados.

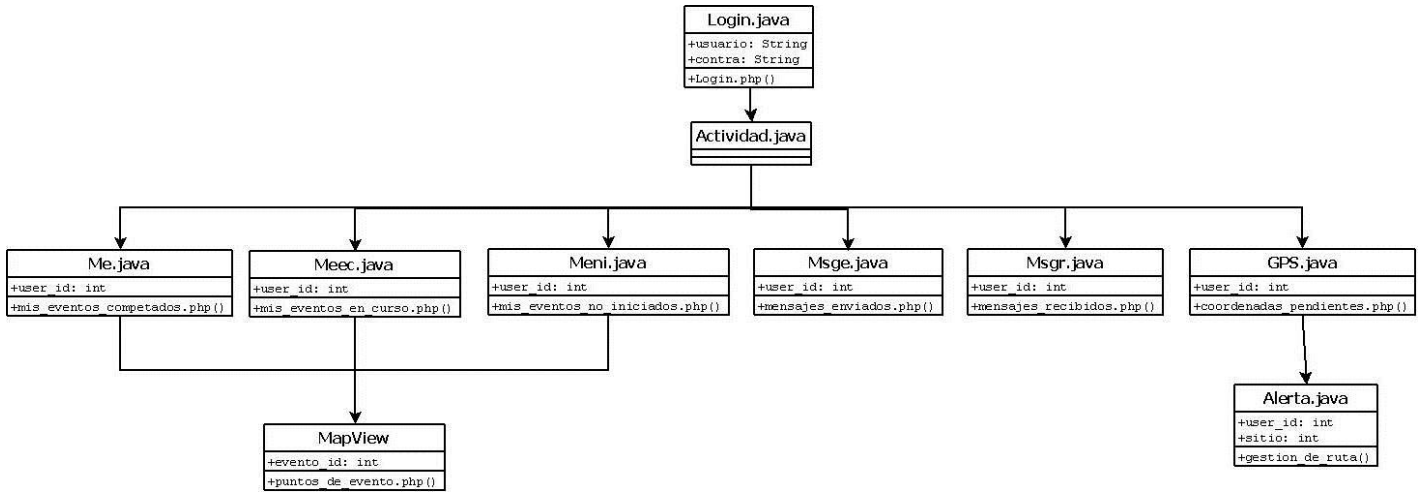
- **Gestión de ruta** Una vez el usuario se ubica en un punto, el Terminal móvil envía la información de que ha llegado a un determinado sitio, enviándolo como parámetro. Posteriormente el controlador comprueba qué punto de la ruta fue visitado anteriormente y en base a eso dictamina si ya se ha estado en ese punto, si se esta procediendo adecuadamente. En caso de haber completado una ruta / evento por completo, el controlador almacenará un "1" en el campo completado de la base de datos como se muestra a continuación:

←T→			id	evento_id	usuario_id	eventos_sitio_id	porcentaje	completado
☐	✎	✕	61	73	1	30	40	0
☐	✎	✕	63	76	1	37	0	1

Figura 3.5 Ejemplo de evento completado

Posteriormente se mostrará por pantalla el mensaje "ruta completada" pasando el evento a ser completado. En el caso de tratarse de un código promocional se mostraría dicho código por pantalla.

A continuación se muestra un diagrama que relaciona los controladores con sus clases, indicando, además, las variables que intercambian en dicho proceso.


Login.java
+usuario: String
+contra: String
+Login.php()
Actividad.java
Me.java
+user_id: int
+mis_eventos_competados.php()
Meec.java
+user_id: int
+mis_eventos_en_curso.php()
Meni.java
+user_id: int
+mis_eventos_no_iniciados.php()
Msge.java
+user_id: int
+mensajes_enviados.php()
Msgr.java
+user_id: int
+mensajes_recibidos.php()
GPS.java
+user_id: int
+coordenadas_pendientes.php()
MapView
+evento_id: int
+puntos_de_evento.php()
Alerta.java
+user_id: int
+sitio: int
+gestion_de_ruta()

3.4 Descripción de la Base de Datos

La base de datos diseñada se llama RutasDB y contiene dieciocho tablas de las cuales siete son usadas por la aplicación móvil y las otras once son solo usadas por la aplicación Web en el servidor.

Estas tablas están relacionadas entre si gracias a las llamadas tablas relacionales que se han nombrado según la nomenclatura descrita por cakePhp ya que de no ser así no funcionaria en el framework y como se busca la compatibilidad se ha optado por seguirlo.

Teniendo en cuenta esto, la tabla que relaciona eventos con sitios se denomina eventos_sitio siendo la "_" el indicador de que estamos ante una tabla que relaciona otras dos tablas; es decir, la tabla relacional. Otro concepto implícito en dicha nomenclatura es la posición de dichas tablas, eventos precede a sitio debido a que son los eventos los que poseen un sitio, lo mismo ocurre con eventos_usuarios; un evento pertenece a uno o varios usuarios y por ultimo mensajes_usuario siendo así ya que un mensaje pertenece a un usuario. De este modo, a la hora de hacer búsquedas con varios parámetros es, no solo posible si no mucho mas sencilla.

Las siete tablas que son de interés son: eventos, usuarios, eventos_sitios, eventos_usuarios, mensajes, mensajes_usuarios y sitios. Serán explicadas a continuación.

Estructura de tabla para la tabla eventos

Campo	Tipo	Nulo	Predeterminado
Id	int(11)	Sí	NULL
desc	varchar(200)	Sí	NULL
Tipo_evento_id	int(11)	Sí	NULL
usuario_id	int(11)	Sí	NULL
eventos_sitio_id	int(11)	Sí	NULL

Tabla 3.1 Tabla eventos.

La tabla eventos almacena los eventos que se van añadiendo desde el servidor, el ampo id es un índice de clave primaria que tiene el valor del campo id del evento registrado, el campo tipo_evento_id indica de qué tipo es el evento con un valor numerico, el campo Desc es un campo de texto, donde se añade el nombre del evento asi como una breve reseña donde ademas se pueden almacenar comentarios relacionados con dicho evento. El campo de usuario_id almacena el identificador del usuario que creo el evento.

Estructura de tabla para la tabla eventos_sitios

Campo	Tipo	Nulo	Predeterminado
id	int(11)	Sí	NULL
evento_id	int(11)	Sí	NULL
sitio_id	int(11)	Sí	NULL

Tabla 3.2 Tabla eventos_sitios.

Esta tabla es una tabla relacional de las tablas eventos y sitios como su propio nombre indica, es decir permite atribuirle un sitio a un evento. Está compuesta de un identificador uno auto-incremental y otros dos campos que sirven propiamente para la relación. Evento_id almacena el id del evento en cuestión haciendo lo mismo el campo sitio_id con su respectivo id.

Estructura de tabla para la tabla eventos_usuarios

Campo	Tipo	Nulo	Predeterminado
Id	int(11)	Sí	NULL
Evento_id	int(11)	Sí	NULL
Usuario_id	int(11)	Sí	NULL
eventos_sitio_id	int(11)	Sí	NULL
porcentaje	Flota	Sí	NULL

completado	int(11)	Sí	NULL
Codigo	varchar(255)	Sí	NULL

Tabla 3.3 Tabla eventos_usuarios.

La tabla eventos_usuarios se compone de un identificador único autoincremental y sirve para atribuir a un usuario la pertenencia de un evento. Es decir, cuando un usuario se apunta a un evento es en esta tabla donde se hace constar. Asimismo, gracias al campo eventos_sitio_id se sabe en qué punto de la ruta en el caso de tener varios sitios un mismo evento se encuentra el usuario, pudiendo calcularse el campo porcentaje, que indica como su propio nombre indica el porcentaje del evento realizado, por ejemplo: si un evento estuviese compuesto de dos sitios, y completamos un sitio este campo se autocompletará con un valor de '50' cuando este valor alcanza el 100% es cuando se marca con un 1 el valor completado que indica que la ruta ya ha sido terminada por completo.

Estructura de tabla para la tabla mensajes

Campo	Tipo	Nulo	Predeterminado
Id	int(11)	Sí	NULL
msg	varchar(255)	Sí	NULL
evento_id	int(11)	Sí	NULL

Tabla 3.4 Tabla Mensajes.

Esta tabla almacena el contenido del mensaje dentro del campo msg y lo relaciona con un id para luego poder ser relacionado en la tabla mensajes_usuarios, que describimos a continuación.

Estructura de tabla para la tabla mensajes_usuarios

Campo	Tipo	Nulo	Predeterminado
Id	int(11)	Sí	NULL
mensaje_id	int(11)	Sí	NULL

udestino_id	int(11)	Sí	NULL
uorigen_id	int(11)	Sí	NULL

Tabla 3.5 Tabla eventos.

Esta tabla relaciona los mensajes con los usuarios, contiene ademas, información sobre el usuario destino y el de origen gracias a los campos udestino_id y uorigen_id respectivamente. El campo mensaje_id nos indica qué id tiene el mensaje en la tabla mensajes.

Estructura de tabla para la tabla sitios

Campo	Tipo	Nulo	Predeterminado
Id	int(11)	Sí	NULL
desc	varchar(200)	Sí	NULL
Lat	Double	Sí	NULL
long	Double	Sí	NULL
direccion	varchar(255)	Sí	NULL

Tabla 3.6 Tabla sitios.

La tabla sitios es donde se almacena toda la información atribuida a un determinado sitio. El campo des nos indica con una cadena de texto una breve reseña sobre el sitio. Los campos lat y long almacenan a su vez las coordenadas geoposicionales del sitio mientras que el campo dirección nos indica la dirección atribuida por Google Maps a esa coordenada.

Estructura de tabla para la tabla usuarios

Campo	Tipo	Nulo	Predeterminado
Id	int(11)	Sí	NULL
nombre	varchar(50)	Sí	NULL

apellidos	varchar(50)	Sí	NULL
Email	varchar(50)	Sí	NULL
Clave	varchar(50)	Sí	NULL
tipo_usuario_id	int(11)	Sí	NULL
facebook_id	bigint(11)	Sí	NULL

Tabla 3.7 Tabla usuarios.

La tabla usuarios contiene toda la información sobre el usuario, utilizándose entre otras cosas para hacer el login de la aplicación móvil. Este login se hace a través del Email y de la clave. Siendo este último valor encriptado en MD5. El campo tipo_usuario_id nos indica de qué tipo es el usuario permitiendo o no determinados accesos y permisos. El campo facebook_id no es usado por la aplicación móvil.

CAPÍTULO 4 CONCLUSIONES

4.1 Conclusiones

En este Proyecto Fin de Carrera se ha desarrollado una aplicación Android para facilitar la interacción del usuario con el mundo real gracias al contenido virtual y los códigos QR, el usuario puede obtener información de su posición y de determinados eventos que se encuentran cerca de su ubicación. Accediendo (de forma transparente para el usuario) a una base de datos que contiene referencias de los sitios que se encuentran alrededor de su posición, y que esta información se actualice conforme vaya avanzando y completando eventos.

El acceso a los datos se realiza mediante una conexión ya sea mediante 3G o WIFI conectando el dispositivo móvil con el servidor, el cual permite acceder a la base de datos MySQL. Para la administración de esta base de datos se han creado unos controladores que son los encargados de obtener las peticiones del usuario y devolver las respuestas de la base de datos.

Se ha complementado esta parte del proyecto con la herramienta de administración Web ubicada en http://pcacribia.upct.es/rutas2/, que se encarga de gestionar todos los eventos y todos los usuarios. Dispone, además, de una interfaz de usuario amigable. Otra gran ventaja de la que esta herramienta de administración dispone es la interacción con Facebook mediante su API, esto permite una mayor difusión entre usuarios, lo que facilita su divulgación.

Las aplicaciones realizadas constan de interfaces intuitivas, atractivas y fáciles de usar, y se han utilizado herramientas de software libre. Habiéndose utilizado herramientas de software libre y comprobado la potencia y funcionalidad de la plataforma Android para el desarrollo de aplicaciones móviles, lo que permite múltiples posibilidades tanto para el desarrollador como para el usuario final.

Por lo tanto, se ha creado un sistema formado por un servidor y un dispositivo móvil smartphone capaz de crear contenidos que den soporte a una interacción entre el mundo real y el virtual, donde además los usuarios pueden interactuar entre ellos haciendo la aplicación más interesante.

Las API DEMO de Google evidencian el potencial que ofrece Android para desarrollar aplicaciones para dispositivos móviles ya que permiten mostrar un ejemplo gráfico antes de implementar código.

Otra característica que ha facilitado mucho el desarrollo ha sido el emulador, de gran utilidad a la hora de depurar aplicaciones y testarlas. El problema que se ha presentado a la hora de la implementación de esta aplicación ha sido la necesidad de hacer pruebas con el GPS. Ahí es cuando el emulador no ha sido capaz, obviamente es imposible que un software pueda disponer de GPS y es por ello que ha hecho falta usar un dispositivo real para probar errores en el código. La posibilidad de conectar el terminal por USB al ordenador e ir probando en tiempo real la aplicación ha sido clave para ahorrar tiempo y facilitar el trabajo por lo que es recomendable el uso de esta práctica.

La integración del dispositivo con el mundo real ha sido simplificada debido a la incorporación del sistema de seguimiento de eventos mediante códigos QR, en esta nueva versión el usuario solo tendrá que escanear dichos códigos para poder optar a todas las opciones que estos ofrecen.

4.2 Líneas Futuras

Se podría implementar la conexión con puntos de acceso (Access Point) dedicados ya que en lugares cerrados el GPS pierde cobertura. Esta implementación tendría la ventaja de almacenar datos específicos de la ubicación donde se encuentren. Por ejemplo, si se tratase de un museo, los puntos de acceso contendrían información detallada de las obras de arte situadas en las salas del museo donde la cobertura fuera insuficiente o nula.

Para futuras implementaciones de código sería importante compilar la aplicación en una versión de Android superior a la 2.1 ya que para entonces todos los terminales Android estarán actualizados. En el caso de este proyecto es un terminal HTC Hero. El motivo de esta actualizar a esta versión es debido a una importante mejora en el Android Market -aplicación de gestión de descarga de aplicaciones- por lo que facilitaría su difusión entre los usuarios con mayor rapidez. Además ello permitiría implementar código basado en una API más elevada, lo que repercute en la potencia de la aplicación.

Otra mejora consistiría en acercar o igualar la funcionalidad del terminal con la del servidor. Por ejemplo, en el móvil no se puede añadir amigos ni crear eventos, pero seria bueno dotarle de estas características con objeto de ofrecer información en tiempo real y desde cualquier lugar a los usuarios.

Implementar un menú de navegación para la aplicación dotándola de funcionalidad extra de control.

Depurar tanto la funcionalidad como el diseño de la aplicación sirviendo como guía sugerencias que hayan sugerido los usuarios.

Dar de alta la aplicación en Android Market para que otros usuarios puedan descargarla sin coste alguno.

Permitir la configuración de otros servidores para ofrecer redundancia de tal manera que si un servidor falla exista la opción de conectarse a otro o para conectarse a servidores que dispongan de información específica.

Definir una opción donde se pueda configurar el radio de alerta del sistema permitiendo modificar la cobertura de las alertas de la aplicación.

CAPÍTULO 5: REFERENCIAS

REFERENCIAS / BIBLIOGRAFIA

- ZXING http://zxing.org
- Ableson F., Collins C. y Sen R. **Android Guía para Desarrolladores,** Ed. Manning 464, Año 2009.

- **Actividad Android** http://developer.android.com/reference/android/app/Activity.html#onCreate%28android.os.Bundle%29

- **Android ¿qué es?** http://developer.android.com/guide/basics/what-is-android.html

- **Android Developers** https://dl-ssl.google.com/android/eclipse/.

- **Android Developers**. http://developer.android.com/index.html.

- **Android SDK** http://developer.android.com/sdk/index.html.

- **Android Wikipedia.** http://es.wikipedia.org/wiki/Android.

- **Android-Spa Foros**. http://www.android-spa.com/index.php.

- **Apache** http://www.apache.org/

- **API DEMOS** Esteban Egea López

- **Ciclo de vida app Android** http://celutron.blogspot.com/2007/12/ciclo-de-vida-de-una-aplicacin-android.html

- **Creación apps Android** http://androide.hijodeblog.com/2009/11/03/creacion-de-aplicaciones-android-parte-2-conceptos-iniciales/

- **Desarrolladores-Android** http://groups.google.com/group/desarrolladores-android?hl=es.

- **Eclipse** http://www.eclipse.org/org/

- **Eclipse Project Download.** http://download.eclipse.org/eclipse/downloads/.

- **HTC.** HERO http://www.htc.com/es/product/tattoo/specification.html.

- **Interfaz Grafica Android** http://developer.android.com/guide/topics/ui/

- **Manual MySQL** http://www.desarrolloweb.com/manuales/34/

- **Manual PHP** http://www.desarrolloweb.com/php/

- **PHP** http://es.wikipedia.org/wiki/PHP

- **PhpMyAdmin** http://es.wikipedia.org/wiki/PhpMyAdmin

- **TFM de Keylon Duran Paulino** TFM de Keylon Duran Paulino-080910-8.pdf

-**TFM de Yobany** Memoria_TFM_Yobany_vFinal2.pdf

CAPÍTULO 6: ANEXOS

6.1 Ejemplo de Creación de una Aplicación en Android

Este apartado pretende mostrar paso a paso la creación de un proyecto Android a modo de manual. Se parte de la base de que Eclipse ya está instalado con el SKD de Android. De no ser así es necesario descargar Eclipse www.eclipse.org/downloads/ y también descargar e instalar el SDK en http://developer.android.com/sdk/index.html. Una vez instalados se puede proceder a la creación de un nuevo proyecto.

Creación de un nuevo proyecto

Para la creación de una aplicación en Android hay que seguir los siguientes pasos que se realizaran desde la ventana de eclipse:

1- Crear el proyecto

Seleccionar la pestaña File -> New -> Android Project, Pulsar el botón Next.

2- Especificar las propiedades

A continuación en la figura 6.1 se muestra una ventana donde es necesario detallar algunas propiedades del proyecto. En concreto, se precisa asignar un valor para:

- **Project name**: Es el nombre del proyecto. En la práctica, será el nombre que reciba la carpeta donde se guardará todo lo relativo al presente proyecto, dentro del workspace. Por ejemplo, "Proyecto".

- **Package name**: Es el nombre del paquete bajo el cual será desarrollado todo el código. Le asignaremos el valor "com.proyecto".

- **Activity name**: Es el nombre de la clase Activity que será creada de forma automática por el plug-in de Eclipse. Esta clase Activity simplemente es una clase ejecutable, capaz de realizar alguna tarea, y es imprescindible en la mayoría de las aplicaciones para Android. Pondremos el nombre "Proyecto".

- **Application name**: Es el nombre de la aplicación que se va a desarrollar. Constituye el nombre visible para el usuario del dispositivo móvil. Por ejemplo, "Proyecto".

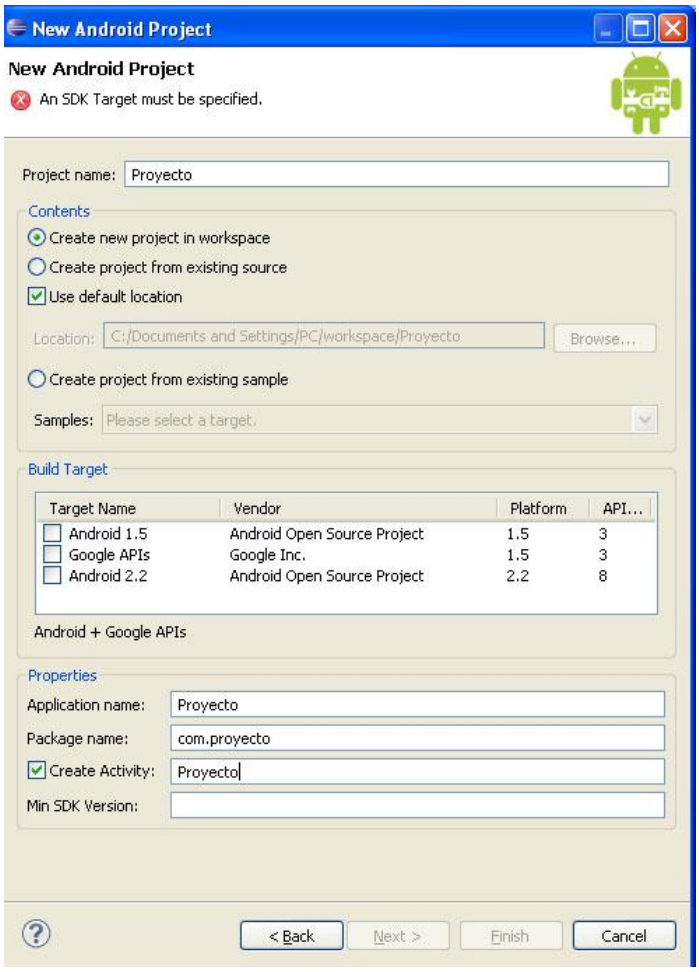


Figura 6.1. Especificaciones para crear proyecto en Android

Tras pulsar el botón Finish, Eclipse mostrará en el explorador de paquetes los elementos que conforman el actual proyecto. Además, si se visita el workspace asignado para los proyectos de Eclipse, podrá observarse que existe una nueva carpeta denominada "Proyecto", ver figura XX

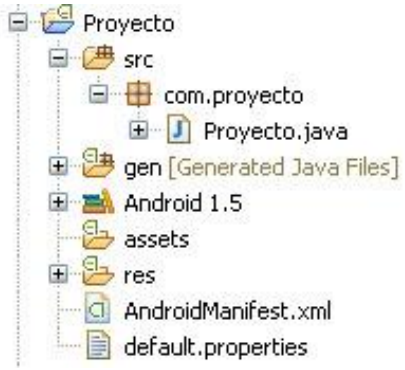


Figura 6.2 Explorador de paquetes

Agregando una interfaz de usuario

Una vez creado el proyecto, si abrimos el archivo "proyecto.java", ubicado en la carpeta "src", tenemos el siguiente código:

```
package com.proyecto;
import android.app.Activity;
import android.os.Bundle;
public class Proyecto extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
}
```

Código 6.1 Contenido de proyecto.java

Al abrir el archivo "main.xml" se puede observar lo siguiente:

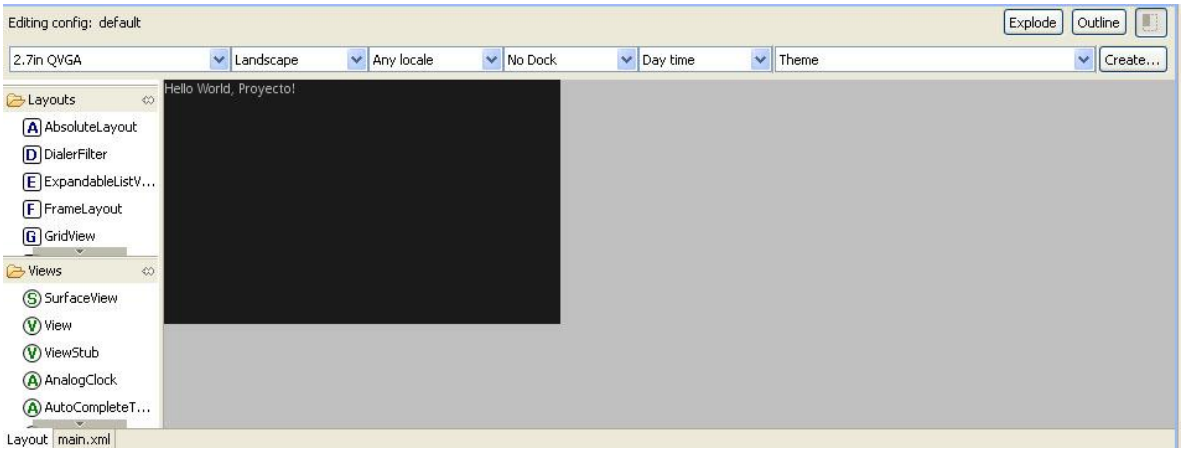


Figura 6.3. Editor gráfico de vistas.

Esto es lo que aparecería en el móvil o en emulador si descargásemos el proyecto. Esta vista grafica responde al siguiente código, que podemos visualizar pulsando en la pestaña que se encuentra en esta misma pantalla en la parte inferior izquierda main.xml.

```
1<?xml version="1.0" encoding="utf-8"?>
2<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3    android:orientation="vertical"
4    android:layout_width="fill_parent"
5    android:layout_height="fill_parent"
6    >
7<TextView
8    android:layout_width="fill_parent"
9    android:layout_height="wrap_content"
10   android:text="@string/hello"
11   />
12</LinearLayout>
13
```

Figura 6.4. Contenido del archivo main.xml

A partir de estas dos vistas es posible modificar y configurar según las necesidades las vistas que luego tendrá el futuro proyecto de Android. Una característica importante que Android ofrece es la posibilidad de modificar el contenido de las vistas mediante código, en el archivo que se encuentra en la carpeta values de nuestro proyecto llamado strings.xml. En este archivo es donde se almacenan todos los datos de tipo String como recurso externo de tal forma que se puede separar la interfaz grafica del código.

Ejecución de la aplicación

Una vez copiado el código anterior hay que descárgalo al emulador o nuestro móvil. En este caso se realiza en el emulador como es una herramienta que trae el SDK de Android para que el usuario vea inmediatamente el resultado del proyecto que hemos creado siguiendo estos pasos:

Lo primero que debemos hacer es seleccionar la pestaña Window->Android SDK and AVD Manager. Desde esta ventana gestionaremos nuestros dispositivos pudiendo agregar nuevos Dispositivos de Android virtual o AVDs.

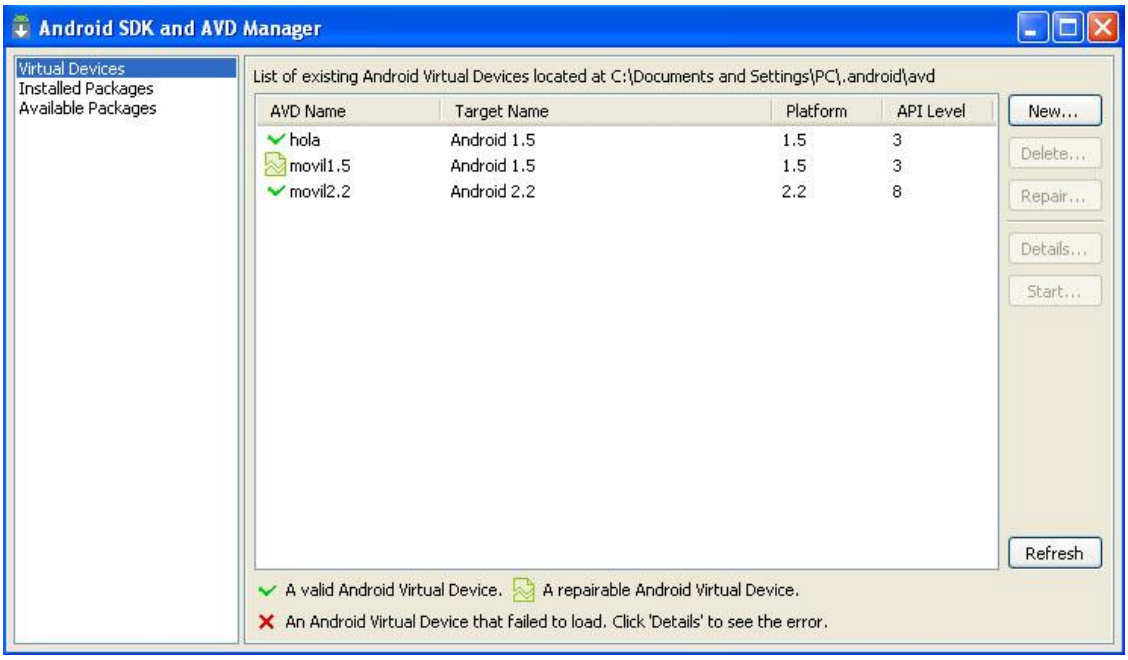


Figura 6.5. Menú para crear un AVD

Solo hay que pulsar el botón New e introducir los datos presentes en la siguiente figura:

Figura 6.6. Especificaciones para crear un AVD

Se selecciona Create AVD para obtener un dispositivo sobre el que ejecutar el código. Acto seguido pulsar sobre la pestaña “run” del menú y seleccionar “run configurations”, aparecerá una ventana tal que esta:

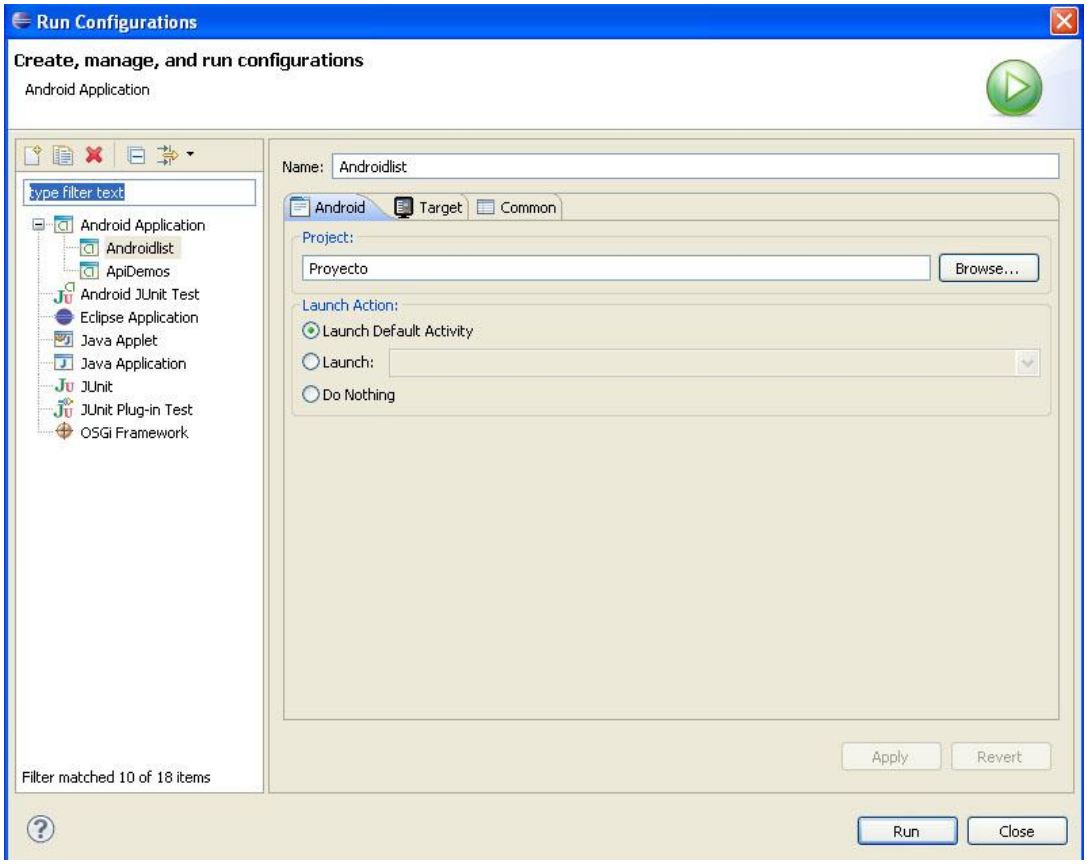


Figura 6.7. Run configurations

Seleccionamos la pestaña donde pone Android y pinchando Browse indicamos el nombre del proyecto que deseamos ejecutar.

Después de haber seleccionado el proyecto estamos listos para seleccionar el dispositivo sobre el que se va a ejecutar el código, para ello es necesario pulsar sobre la pestaña Target, y marcar el radioboton de Automatic seleccionando el dispositivo que se acaba de crear.

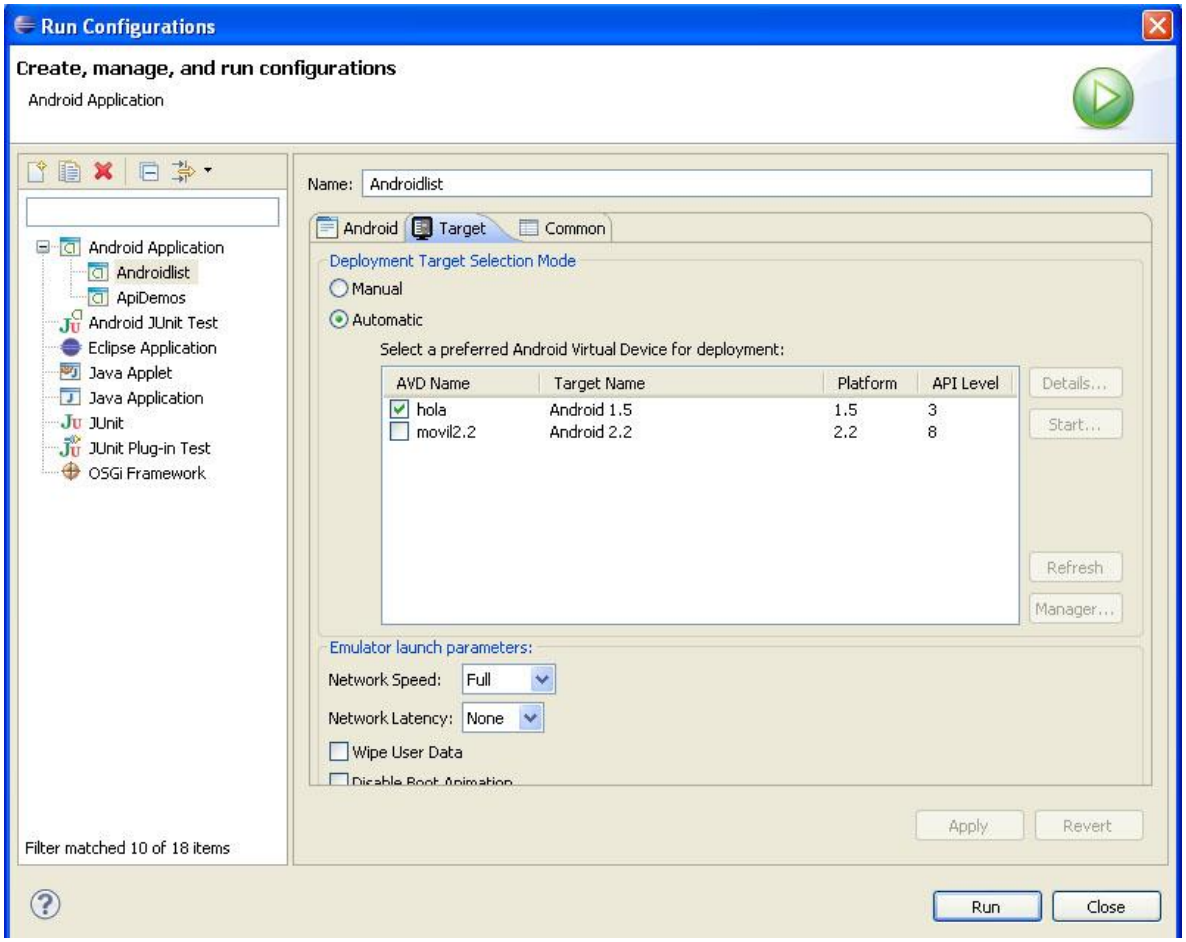


Figura 6.8. Pantalla de selección del target sobre el que se va a ejecutar la aplicación

6.2 Clases IntentIntegrator e IntentResult

Clase IntentIntegrator:

```
package com.google.zxing.integration.android;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.ActivityNotFoundException;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.pm.ResolveInfo;
import android.net.Uri;
import android.os.Bundle;
public class IntentIntegrator {
  public static final int REQUEST_CODE = 0x0000c0de; // Only use bottom 16
bits
```

```
  private static final String TAG = IntentIntegrator.class.getSimpleName();
  public static final String DEFAULT_TITLE = "Install Barcode Scanner?";
  public static final String DEFAULT_MESSAGE =
      "This application requires Barcode Scanner. Would you like to install
it?";
  public static final String DEFAULT_YES = "Yes";
  public static final String DEFAULT_NO = "No";
  private static final String BS_PACKAGE = "com.google.zxing.client.android";
  private static final String BSPLUS_PACKAGE = "com.srowen.bs.android";
  // supported barcode formats
  public static final Collection<String> PRODUCT_CODE_TYPES = list("UPC_A",
"UPC_E", "EAN_8", "EAN_13", "RSS_14");
  public static final Collection<String> ONE_D_CODE_TYPES =
      list("UPC_A", "UPC_E", "EAN_8", "EAN_13", "CODE_39", "CODE_93",
"CODE_128",
           "ITF", "RSS_14", "RSS_EXPANDED");
  public static final Collection<String> QR_CODE_TYPES =
Collections.singleton("QR_CODE");
  public static final Collection<String> DATA_MATRIX_TYPES =
Collections.singleton("DATA_MATRIX");
  public static final Collection<String> ALL_CODE_TYPES = null;

  public static final List<String> TARGET_BARCODE_SCANNER_ONLY =
Collections.singletonList(BS_PACKAGE);
  public static final List<String> TARGET_ALL_KNOWN = list(
          BSPLUS_PACKAGE,              // Barcode Scanner+
          BSPLUS_PACKAGE + ".simple", // Barcode Scanner+ Simple
          BS_PACKAGE                   // Barcode Scanner
          // What else supports this intent?
      );

  private final Activity activity;
  private String title;
  private String message;
  private String buttonYes;
  private String buttonNo;
  private List<String> targetApplications;
  private final Map<String,Object> moreExtras;

  public IntentIntegrator(Activity activity) {
    this.activity = activity;
    title = DEFAULT_TITLE;
    message = DEFAULT_MESSAGE;
    buttonYes = DEFAULT_YES;
    buttonNo = DEFAULT_NO;
    targetApplications = TARGET_ALL_KNOWN;
    moreExtras = new HashMap<String,Object>(3);
  }

  public String getTitle() {
    return title;
  }

  public void setTitle(String title) {
    this.title = title;
  }
  public void setTitleByID(int titleID) {
    title = activity.getString(titleID);
  }
  public String getMessage() {
    return message;
  }
  public void setMessage(String message) {
    this.message = message;
  }
  public void setMessageByID(int messageID) {
```

```
    message = activity.getString(messageID);
  }
  public String getButtonYes() {
    return buttonYes;
  }
  public void setButtonYes(String buttonYes) {
    this.buttonYes = buttonYes;
  }
  public void setButtonYesByID(int buttonYesID) {
    buttonYes = activity.getString(buttonYesID);
  }
  public String getButtonNo() {
    return buttonNo;
  }
  public void setButtonNo(String buttonNo) {
    this.buttonNo = buttonNo;
  }
  public void setButtonNoByID(int buttonNoID) {
    buttonNo = activity.getString(buttonNoID);
  }

  public Collection<String> getTargetApplications() {
    return targetApplications;
  }

  public final void setTargetApplications(List<String> targetApplications) {
    if (targetApplications.isEmpty()) {
      throw new IllegalArgumentException("No target applications");
    }
    this.targetApplications = targetApplications;
  }

  public void setSingleTargetApplication(String targetApplication) {
    this.targetApplications = Collections.singletonList(targetApplication);
  }
  public Map<String,?> getMoreExtras() {
    return moreExtras;
  }
  public final void addExtra(String key, Object value) {
    moreExtras.put(key, value);
  }
  /**
   * Initiates a scan for all known barcode types.
   */
  public final AlertDialog initiateScan() {
    return initiateScan(ALL_CODE_TYPES);
  }

  public final AlertDialog initiateScan(Collection<String>
desiredBarcodeFormats) {
    Intent intentScan = new Intent(BS_PACKAGE + ".SCAN");
    intentScan.addCategory(Intent.CATEGORY_DEFAULT);
    // check which types of codes to scan for
    if (desiredBarcodeFormats != null) {
      // set the desired barcode types
      StringBuilder joinedByComma = new StringBuilder();
      for (String format : desiredBarcodeFormats) {
        if (joinedByComma.length() > 0) {
          joinedByComma.append(',');
        }
        joinedByComma.append(format);
      }
      intentScan.putExtra("SCAN_FORMATS", joinedByComma.toString());
    }
    String targetAppPackage = findTargetAppPackage(intentScan);
    if (targetAppPackage == null) {
```

```
      return showDownloadDialog();
    }
    intentScan.setPackage(targetAppPackage);
    intentScan.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    intentScan.addFlags(Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);
    attachMoreExtras(intentScan);
    startActivityForResult(intentScan, REQUEST_CODE);
    return null;
  }
  protected void startActivityForResult(Intent intent, int code) {
    activity.startActivityForResult(intent, code);
  }

  private String findTargetAppPackage(Intent intent) {
    PackageManager pm = activity.getPackageManager();
    List<ResolveInfo> availableApps = pm.queryIntentActivities(intent,
PackageManager.MATCH_DEFAULT_ONLY);
    if (availableApps != null) {
      for (String targetApp : targetApplications) {
        if (contains(availableApps, targetApp)) {
          return targetApp;
        }
      }
    }
    return null;
  }

  private static boolean contains(Iterable<ResolveInfo> availableApps, String
targetApp) {
    for (ResolveInfo availableApp : availableApps) {
      String packageName = availableApp.activityInfo.packageName;
      if (targetApp.equals(packageName)) {
        return true;
      }
    }
    return false;
  }
  private AlertDialog showDownloadDialog() {
    AlertDialog.Builder downloadDialog = new AlertDialog.Builder(activity);
    downloadDialog.setTitle(title);
    downloadDialog.setMessage(message);
    downloadDialog.setPositiveButton(buttonYes, new
DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialogInterface, int i) {
        String packageName = targetApplications.get(0);
        Uri uri = Uri.parse("market://details?id=" + packageName);
        Intent intent = new Intent(Intent.ACTION_VIEW, uri);
        try {
          activity.startActivity(intent);
        } catch (ActivityNotFoundException anfe) {
          // Hmm, market is not installed
          Log.w(TAG, "Google Play is not installed; cannot install " +
packageName);
        }
      }
    });
    downloadDialog.setNegativeButton(buttonNo, new
DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialogInterface, int i) {}
    });
    return downloadDialog.show();
  }
  public static IntentResult parseActivityResult(int requestCode, int
resultCode, Intent intent) {
```

```
    if (requestCode == REQUEST_CODE) {
      if (resultCode == Activity.RESULT_OK) {
        String contents = intent.getStringExtra("SCAN_RESULT");
        String formatName = intent.getStringExtra("SCAN_RESULT_FORMAT");
        byte[] rawBytes = intent.getByteArrayExtra("SCAN_RESULT_BYTES");
        int intentOrientation = intent.getIntExtra("SCAN_RESULT_ORIENTATION",
Integer.MIN_VALUE);
        Integer orientation = intentOrientation == Integer.MIN_VALUE ? null :
intentOrientation;
        String errorCorrectionLevel =
intent.getStringExtra("SCAN_RESULT_ERROR_CORRECTION_LEVEL");
        return new IntentResult(contents,
                                formatName,
                                rawBytes,
                                orientation,
                                errorCorrectionLevel);
      }
      return new IntentResult();
    }
    return null;
  }
  public final AlertDialog shareText(CharSequence text) {
    return shareText(text, "TEXT_TYPE");
  }
  public final AlertDialog shareText(CharSequence text, CharSequence type) {
    Intent intent = new Intent();
    intent.addCategory(Intent.CATEGORY_DEFAULT);
    intent.setAction(BS_PACKAGE + ".ENCODE");
    intent.putExtra("ENCODE_TYPE", type);
    intent.putExtra("ENCODE_DATA", text);
    String targetAppPackage = findTargetAppPackage(intent);
    if (targetAppPackage == null) {
      return showDownloadDialog();
    }
    intent.setPackage(targetAppPackage);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);
    attachMoreExtras(intent);
    activity.startActivity(intent);
    return null;
  }

  private static List<String> list(String... values) {
    return Collections.unmodifiableList(Arrays.asList(values));
  }
  private void attachMoreExtras(Intent intent) {
    for (Map.Entry<String,Object> entry : moreExtras.entrySet()) {
      String key = entry.getKey();
      Object value = entry.getValue();
      // Kind of hacky
      if (value instanceof Integer) {
        intent.putExtra(key, (Integer) value);
      } else if (value instanceof Long) {
        intent.putExtra(key, (Long) value);
      } else if (value instanceof Boolean) {
        intent.putExtra(key, (Boolean) value);
      } else if (value instanceof Double) {
        intent.putExtra(key, (Double) value);
      } else if (value instanceof Float) {
        intent.putExtra(key, (Float) value);
      } else if (value instanceof Bundle) {
        intent.putExtra(key, (Bundle) value);
      } else {
        intent.putExtra(key, value.toString());
      }
    }
```

```
  }
}
```

Clase IntentResult:

```
package com.google.zxing.integration.android;
public final class IntentResult {
  private final String contents;
  private final String formatName;
  private final byte[] rawBytes;
  private final Integer orientation;
  private final String errorCorrectionLevel;
  IntentResult() {
    this(null, null, null, null, null);
  }
  IntentResult(String contents,
               String formatName,
               byte[] rawBytes,
               Integer orientation,
               String errorCorrectionLevel) {
    this.contents = contents;
    this.formatName = formatName;
    this.rawBytes = rawBytes;
    this.orientation = orientation;
    this.errorCorrectionLevel = errorCorrectionLevel;
  }
  /**
   * @return raw content of barcode
   */
  public String getContents() {
    return contents;
  }
  /**
   * @return name of format, like "QR_CODE", "UPC_A". See {@code
BarcodeFormat} for more format names.
   */
  public String getFormatName() {
    return formatName;
  }
  /**
   * @return raw bytes of the barcode content, if applicable, or null
otherwise
   */
  public byte[] getRawBytes() {
    return rawBytes;
  }
  /**
   * @return rotation of the image, in degrees, which resulted in a successful
scan. May be null.
   */
  public Integer getOrientation() {
    return orientation;
  }
  /**
   * @return name of the error correction level used in the barcode, if
applicable
   */
  public String getErrorCorrectionLevel() {
    return errorCorrectionLevel;
  }

  @Override
  public String toString() {
    StringBuilder dialogText = new StringBuilder(100);
```

```
    dialogText.append("Format: ").append(formatName).append('\n');
    dialogText.append("Contents: ").append(contents).append('\n');
    int rawBytesLength = rawBytes == null ? 0 : rawBytes.length;
    dialogText.append("Raw bytes: (").append(rawBytesLength).append("
bytes)\n");
    dialogText.append("Orientation: ").append(orientation).append('\n');
    dialogText.append("EC level: ").append(errorCorrectionLevel).append('\n');
    return dialogText.toString();
  }
}
```

6.3 Características técnicas del HTC Hero

Figura 6.9. Terminal HTC HERO

General

Redes: GSM 850 / 900 / 1800 / 1900 – HSDPA 900 / 2100 | HSDPA 850 / 1900
Sistema Operativo Android OS, v1.5
Tamaño del HTC Hero
Dimensiones 112 x 56.2 x 14.4 mm
Peso 135 g
Display
Tipo TFT touchscreen capacitivo, 65K colores
Tamaño 320 x 480 pixels, 3.2 pulgadas
- Método de ingreso de datos multi táctil
- Sensor acelerómetro para auto rotación
- Reconocimiento escritura de mano
- Trackball
Ringtones Polifónico, MP3
Vibrador
Conector de audio Jack 3.5 mm
Slot de tarjeta microSD (TransFlash)
- 288 MB RAM, 512 MB ROM memoria interna
- Procesador Qualcomm MSM 7200A 528 MHz
Características GPRS clase 10 (4+1/3+2 slots)
Velocidad de datos 32 – 48 kbps
Mensajería SMS, MMS, Email, Mensajería instantánea

Navegador HTML
Juegos
Colores Marrón, Blanco (protección con teflón)
Cámara de fotos y video de 5 Megapíxeles, 2592 x 1944 pixels, Autoefoque
GPS con soporte A-GPS
Brújula digital
HSCSD
EDGE Clase 10
Datos e Internet 3G HSDPA/HSUPA
Wi-Fi
Bluetooth A2DP
Mini USB
Tecla de búsqueda dedicada
Reproductor MP3/AAC+/WAV/WMA9
Reproductor MPEG-4/H.263/H.264/WMV9
Manos libres incorporado
Aplicaciones Google
Memo de voz
Batería Standard, Li-Ion 1350 mAh
Stand-by Hasta 440 h (2G) / Hasta 750 h (3G)
Tiempo de conversación Hasta 8 h (2G) / Hasta 7 h (3G)