UNIVERSIDAD POLITECNICA DE VALENCIA

UNIVERSIDAD POLITÉCNICA DE VALENCIA
ESCUELA TÉCNICA SUPERIOR DE INFORMÁTICA APLICADA

Aplicación para Android: agenda comercial y detalles de clientes

PROYECTO FIN DE CARRERA

Autor
Ignacio Domingo Garzarán

Directores
Carlos Tavares Calafate
Emma López Franco

Valencia, a 9 de septiembre de 2010

Resumen

Este proyecto final de carrera consiste en la creación de una aplicación para *Android* alimentada por peticiones *REST* (*Representational State Transfer*) sobre *Visual Studio 2008* en *C#*. Se parte de una serie de librerías ya desarrolladas que sirven información sobre una aplicación que tiene usuarios (gestores) que tienen clientes, y dónde éstos a su vez tienen carteras de inversión. El gestor tiene una agenda de tareas y citas, así como alertas creadas sobre clientes y productos.

La aplicación debe ser capaz de:

1. Realizar una búsqueda de clientes y mostrar el resultado del mismo
2. Ver los datos personales de sus clientes
3. Ver el calendario de tareas y de citas general del usuario
4. Ver el calendario de tareas y citas asociado a un cliente dado
5. Dar de alta tareas/citas asociadas a un cliente dado
6. Ver las Alertas de Productos
7. Ver las Alertas de Clientes
8. Ver un informe de las carteras de cliente:
    - Importe de la cartera por producto
    - Distribución de los productos

PALABRAS CLAVE

Android, aplicación móvil, Workplace, agenda comercial

Índice

1 Introducción

Este documento tiene como propósito detallar la labor realizada durante el periodo de ejecución del proyecto final de carrera, el cual consiste de una aplicación para dispositivos móviles *Android* y ha sido desarrollado en la empresa *Openfinance S.L*.

La aplicación es una agenda comercial que contacta con el *Workplace* de la empresa (consultar anexo para más información) a través de servicios web REST (*Representational State Transfer*). Esto supondrá diversas ventajas tanto para la empresa como para los usuarios ya existentes del *Workplace*.

Por un lado, la empresa ampliará su gama de productos, ofreciendo a sus usuarios del *Workplace* una nueva forma de acceder a sus datos. Además, al aparecer la aplicación en el *Android Market*, siempre existe la posibilidad de captar nuevos clientes.

Poro otro lado, los usuarios podrán consultar sus datos en cualquier momento y lugar, ya que todos los dispositivos *Android* disponen de acceso a Internet. Además esta aplicación añade nuevas funcionalidades para interactuar con sus datos

Puesto que el objetivo de realizar un proyecto fin de carrera no es sólo la aplicación en sí, sino el paso de estudiante a ingeniero, la estructura del documento sigue una línea temporal, dónde se explica al inicio del mismo los pasos previos que cualquier Ingeniero Informático debe realizar antes de comenzar la implementación, y concluyendo con las ideas y reflexiones obtenidas tras su finalización, tanto sobre la aplicación como a nivel personal en la etapa final como estudiante.

Como último aspecto importante, este trabajo ha servido como primera toma de contacto con el mundo laboral, además de como finalización del ciclo de formación universitaria.

1.1 Motivación

La finalidad de este proyecto es ofrecer a los clientes ya existentes de *Openfinance* nuevas formas de acceder a sus aplicaciones. Por una parte, esto aporta beneficios a la empresa puesto que sus productos no se quedan obsoletos y se adaptan a las nuevas tecnologías. Por otra parte, los clientes ven como la empresa evoluciona y les proporciona nuevas formas de acceso a sus datos; esto proporcionará confianza y satisfacción de los usuarios actuales, al permitir comprobar que el *Workplace* es capaz de adaptarse en cada momento a las exigencias del mercado.

Al mismo tiempo, esta aplicación puede ayudar a la captación de nuevos clientes, bien sea por descubrimiento de la existencia de la empresa al acceder a esta aplicación desde su dispositivo *Android*, o clientes que utilizaban productos similares al *Workplace* y que pueden sentirse atraídos por las nuevas posibilidades que este proyecto les ofrece, ya que es posible que las herramientas que estén utilizando no se hayan adaptado a las nuevas tecnologías.

Existe ya una amplia gama de aplicaciones sobre economía para dispositivo móviles *Android,* pero ninguna similar a una intranet a la que puedes acceder desde tu teléfono. Ésta en concreto está orientada a usuarios gestores, y las ventajas que aporta respecto a otras aplicaciones de gestión de inversiones, es que con esta aplicación no solo podrán mantener un seguimiento de las carteras de inversión de sus clientes desde cualquier lugar, sino que además le ofrece una agenda comercial completa que le permitirá ponerse en contacto con sus clientes de múltiples formas.

De esta manera, las ventajas que aporta esta aplicación respecto a las ya existentes sobre finanzas son todas las características definidas en el Anexo sobre el Workplace. Todo ello hace que, a nivel nacional, sea la empresa líder en el desarrollo de aplicaciones de gestión.

1.2 Objetivos

Hoy en día es cada vez más habitual el uso de Internet en un dispositivo móvil. En el caso de *Android*, y más teniendo en cuenta que el principal promotor de este sistema operativo es *Google*, los dispositivos móviles están orientados a un uso optimizado de Internet.

Teniendo en cuenta las nuevas tendencias del mercado, el principal objetivo de este proyecto es proporcionar una nueva forma de acceso a los usuarios de Workplace, que es una herramienta de gestión que básicamente permite al usuario realizar un seguimiento de las inversiones de sus clientes. Concretamente, la versión inicial se ha implementado para hacerla funcionar sobre el *Workplace* de *EAFIS* (definición en el apartado de *Anexos*).

Esta aplicación logra que un usuario pueda acceder e interactuar con sus datos de una manera rápida y sencilla desde cualquier lugar en el que su dispositivo móvil tenga cobertura. De esta manera, este objetivo principal puede ser dividido en dos bloques:

- Acceso eficiente a sus datos: presentárselos de una manera ordenada buscando siempre la optimización de los recursos disponibles. Para ello la aplicación estará dividida en tres menús: de usuario, de cliente y de carteras de un cliente.

- Interacción de sus datos con la plataforma en la que accede a ellos: gracias a esta aplicación no sólo podrá acceder a sus datos para verlos, sino que además podrá realizar llamadas, enviar mensajes de texto o correos electrónicos.

Respecto a los objetivos personales, teniendo en cuenta que el mundo de la informática está en continua evolución, se busca aprender a utilizar las nuevas tecnologías puesto que aunque se termine la etapa de estudiante, un informático va a estar aprendiendo a usar nuevos lenguajes de programación o tecnologías durante toda su vida laboral.

En este caso, el objetivo personal es aprender a programar para una plataforma en la que no se ha trabajado durante la carrera, dado que *Android* es un sistema operativo muy reciente en el que aún no se ha profundizado a nivel académico.

1.3 Estructura del documento

Como ya se comentaba en la introducción, el presente documento está organizado de forma que sigue la línea temporal del proyecto. Cada uno de sus apartados define una etapa del proceso.

La primera etapa, descrita en la sección 2, es el análisis y el diseño. Ésta debe ser común a cualquier proyecto a realizar y el paso previo a la implementación ya que debe quedar muy claro qué puede hacer la aplicación, quienes lo van a hacer y cómo se podrá realizar. En este caso esta etapa fue muy breve, puesto que al contar la empresa con una aplicación similar para *Iphone* y los servicios web ya desarrollados, los requisitos y especificaciones quedaban ya muy claros. De esta manera, en este apartado se explica, además de los respectivos diagramas y especificación de requisitos, la forma de acceder y tratar los datos mediante los servicios web.

Una vez ha quedado claro qué debe hacerse y cómo hacerlo, en la sección 3 se explica cómo se ha realizado el proceso de implementación, pero únicamente a nivel de organización puesto que una correcta planificación debe hacerse siempre independientemente de las tecnologías y lenguajes empleados. Los detalles se explican al final cuando se haya visto el proceso en su conjunto.

En la sección 4 lo que se explica es el proceso de validación y pruebas, que en verdad se realiza a la vez que la implementación, pero para comprender su relación con esta fase es mejor ver antes como se ha organizado el proceso de desarrollo. Además, en este caso al ser una aplicación para dispositivos móviles, es más conveniente realizar la validación y pruebas finales en un dispositivo real cuando la aplicación está prácticamente terminada.

Tras ver todo el proceso organizado temporalmente, en la sección 5 se procede a explicar de forma más detallada como se ha realizado la implementación y que tecnologías se han utilizado para ello, puesto que este periodo es el más largo de todos.

Después de explicar cómo se ha implementado la aplicación, la sección 6 muestra los resultados de todas estas fases con una breve descripción de estos resultados obtenidos.

Casi unidos a este apartado, los dos siguientes y últimos apartados de la línea temporal muestran las ideas y reflexiones obtenidas tras tener la aplicación totalmente finalizada y en funcionamiento. El primero de ellos (sección 7), más relacionado con la aplicación en sí, describe aspectos que podrían mejorarse o añadir. En el último aparatado (sección 8) ya se expresan las conclusiones referidas a toda la etapa del proyecto.

2 Análisis y diseño

Todo proyecto debe comenzarse con un correcto estudio previo a la implementación, que defina claramente que acciones se podrán realizar, quién las podrá hacer y cómo va a funcionar, sin entrar en detalles de implementación.

Así siendo, en este apartado se muestran una serie de pasos previos a la realización del código del programa que ayudarán a comprender, elaborar y mantener correctamente la aplicación posteriormente.

2.1 Especificación de requisitos

Al comienzo de este proyecto la empresa *Openfinance* ya disponía de una aplicación similar para dispositivos móviles *iPhone,* con lo cual todas las decisiones de apariencia de la aplicación y requisitos funcionales estaban ya preestablecidos. El objetivo era que el resultado final fuese similar a la aplicación para *iPhone,* pero a su vez, adaptándose a las interfaces de usuario habituales en aplicaciones para *Android*.

De esta manera, los requisitos establecidos finalmente son los siguientes.

- Autenticación: el usuario deberá autenticarse con sus datos del *Workplace* (usuario y contraseña).

- Seguridad: una vez autenticado el usuario, este podrá elegir salir de la aplicación haciendo *logout* o por el contrario, que sus datos se almacenen y no sea necesario introducirlos la próxima vez que abra la aplicación. Para proteger el acceso a sus datos, el usuario puede determinar que cada vez que inicie la aplicación se solicite un PIN de cuatro dígitos.

- Interfaz y usabilidad: debe ser clara y sencilla. De apariencia similar a los menús propios de la plataforma *Android*.

- Navegación en las pantallas de la aplicación: debe ser distinguible cuando se encuentra en pantallas de información de usuario y cuando en ventanas con información del cliente. Esta distinción se hará mediante las pestañas correspondientes al panel de cada menú.

- Integración en el dispositivo: puesto que la información a consultar se presenta en un dispositivo móvil, esta deberá poder interactuar con las posibilidades que el terminal ofrece siempre que sea posible: realizar llamadas, enviar SMS o correo electrónico.

– Ampliación: la aplicación debe ser adaptable a futuras modificaciones o ampliaciones.

2.2 Especificaciones técnicas

Openfinance cuenta con una serie de servicios web REST que proporcionarán la información con la que la aplicación trabaja. Esto hace que la arquitectura de la aplicación y su manera de comunicarse con el servidor ya esté determinada al inicio del proyecto. En la Figura 1 se muestra un esquema que explica cómo debe funcionar el programa.

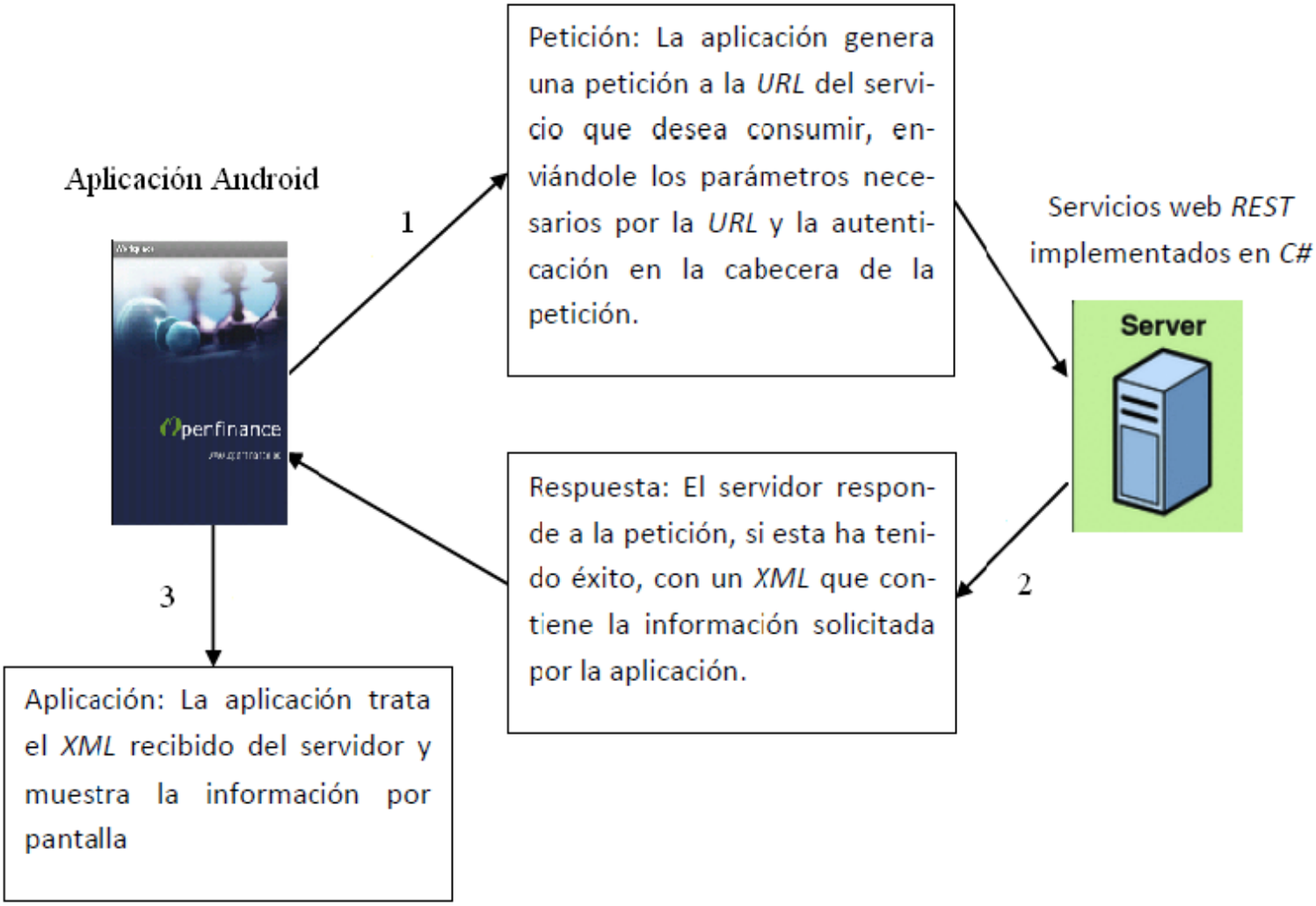


Figura 1: esquema de la aplicación

2.2.1 Autenticación HTTP Basic

Para establecer la conexión con el servidor y proteger los servicios web se debe utilizar la autenticación HTTP Basic.

Este tipo de autenticación es simple y es el más usado para proteger los recursos. Si se hace la petición desde un navegador, este utiliza una ventana de diálogo propia que solicita el nombre de usuario y contraseña; en este caso esto deberá enviarse como cabecera en el momento de realizar la conexión. El problema es que no cifra los datos; en todas las peticiones que se realizan se envía tanto el usuario como la contraseña sin cifrar en la petición, siendo la codificación Base64 el único paso previo a la conexión. La forma de implementar este proceso se explicará más adelante.

2.2.2 Conexión REST

La idea de *REST* (*Representational State Transfer*) es aprovechar las capacidades del protocolo *HTTP* para utilizarlo como arquitectura de servicios. *REST* describe una interfaz web simple utilizando peticiones *HTTP* y datos *XML* pero sin capas adicionales como *SOAP*. Actualmente es la alternativa más simple a *SOAP* y a los servicios web basados en *WSDL* (*Web Services Descripcion Language*).

Estos servicios web son particularmente útiles en dispositivos con recursos escasos como *PDAs* o teléfonos móviles, donde la sobrecarga de las cabeceras y capas adicionales de los elementos *SOAP* debe ser restringida.

REST no posee un modo estándar y formal de describir la interfaz de los servicios web, por lo que ambas partes deben estar de acuerdo en el modo de intercambiar la información. En este caso el servidor ya está creado y la aplicación cliente para *Android* debe construirse de manera a mantener la compatibilidad con los ficheros que éste devuelve.

En la Figura 2, se puede observar un esquema que representa este protocolo. En el esquema se ve como hay un paquete *HTTP*, con los parámetros que necesita en la cabecera y el cuerpo, que va del cliente al servicio o del servicio al cliente.

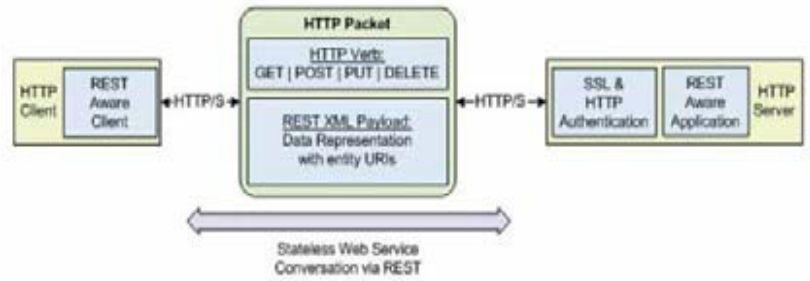


Figura 2: comunicación REST

La conexión con el servidor debe volver a realizarse en cada petición, enviando de nuevo nombre de usuario y contraseña puesto que cada mensaje *HTTP* contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.

Una petición completa e independiente hace que el servidor no tenga que recuperar ninguna información de contexto o estado al procesar la petición. Una aplicación o cliente de servicio web

REST debe incluir dentro del encabezado y del cuerpo *HTTP* de la petición todos los parámetros, contexto y datos que necesita el servidor para generar la respuesta. De esta manera, el no mantener estado mejora el rendimiento de los servicios web, ya que la ausencia de estado en el servidor elimina la necesidad de sincronizar los datos de la sesión con una aplicación externa.

Un servicio sin estado no sólo funciona mejor, sino que además evita la responsabilidad de mantener el estado al cliente de la aplicación. En estos servicios web, el servidor es el responsable de generar las respuestas y proveer una interfaz que le permita al cliente mantener el estado de la aplicación por su cuenta.

Otra característica importante es que permite la transferencia de los datos en formato *XML*, JSON o de ambos. En este caso sólo se devuelven datos *XML*, pero una ampliación importante a tener en cuenta es la posibilidad de que también pueda devolverse en formato *JSON*, puesto que el uso de los tipos *MIME* y del encabezado *HTTP Accept* es un mecanismo conocido como *negociación de contenido*, el cual le permite a los clientes elegir qué formato de datos puedan leer, y minimiza el acoplamiento de datos entre el servicio y las aplicaciones que lo consumen.

Teniendo en cuenta este último aspecto, la aplicación debe implementarse de manera que sea fácilmente adaptable a una futura incorporación de servicios web que devuelvan los datos en notación *JSON*. Es decir, que sea totalmente independiente en el código la forma de tratar los datos a como se accede a ellos.

2.2.3 Información intercambiada

Como ya se ha comentado, el intercambio de información se realiza a través de ficheros *XML*. Estas son algunas de las ventajas que proporciona este formato de datos:

- Permite la definición de datos de manera preciosa, puesto que se pueden definir tipos de datos y cardinalidad.

- Aporta tipos de datos nativos, los cuales son comunes a todos los lenguajes de programación: *int, String, float, date*...

- La jerarquía de elementos, permite agrupar diversos atributos lo cual es muy útil, ya que se puede modularizar la especificación del *XML* lo que lo hace mucho más fácil y cómodo de entender.

- Es extensible, se puede ampliar añadiendo nuevas etiquetas sin comprometer el correcto funcionamiento de versiones anteriores.

- Se puede trabajar con varios espacios de nombres.

2.3 Diagramas UML

Tras la fase de análisis, se realizan los correspondientes diagramas que permiten diseñar la aplicación. Con ello se intenta mostrar como funcionará la aplicación, cuál es su flujo de datos o como están organizadas las clases que representan los datos del *Workplace*.

2.3.1 Diagrama de casos de uso

En este diagrama se muestran todas las acciones que el usuario puede realizar para interactuar con el sistema. Para realizar cualquiera de ellas debe haberse autenticado previamente.

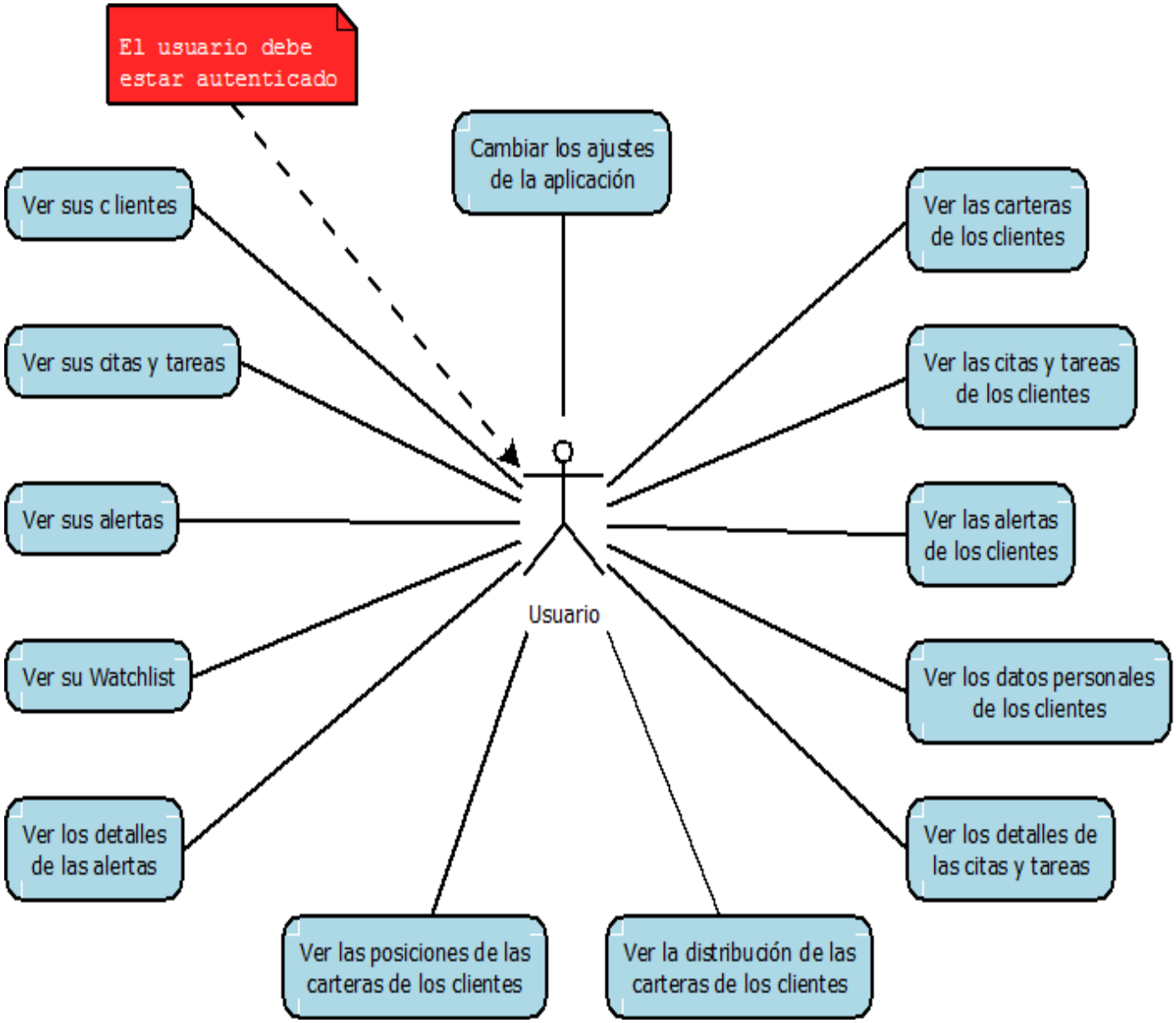


Figura 3: Diagrama de casos de uso

2.3.2 Diagrama de actividades

En este diagrama se muestra en el orden en el que el usuario debe realizar las acciones, así como las condiciones que deben cumplirse para poder llevarlas a cabo. El estado final puede alcanzarse desde cualquier actividad. La autenticación se refiere tanto al *login* como al PIN, puesto que, según la voluntad del usuario al iniciarse la aplicación se pueden requerir ambas, una de ellas o ninguna. Para una mayor claridad el diagrama se encuentra dividido en esta página (en la que se muestran sólo las actividades referidas al menú del cliente) y la siguiente.

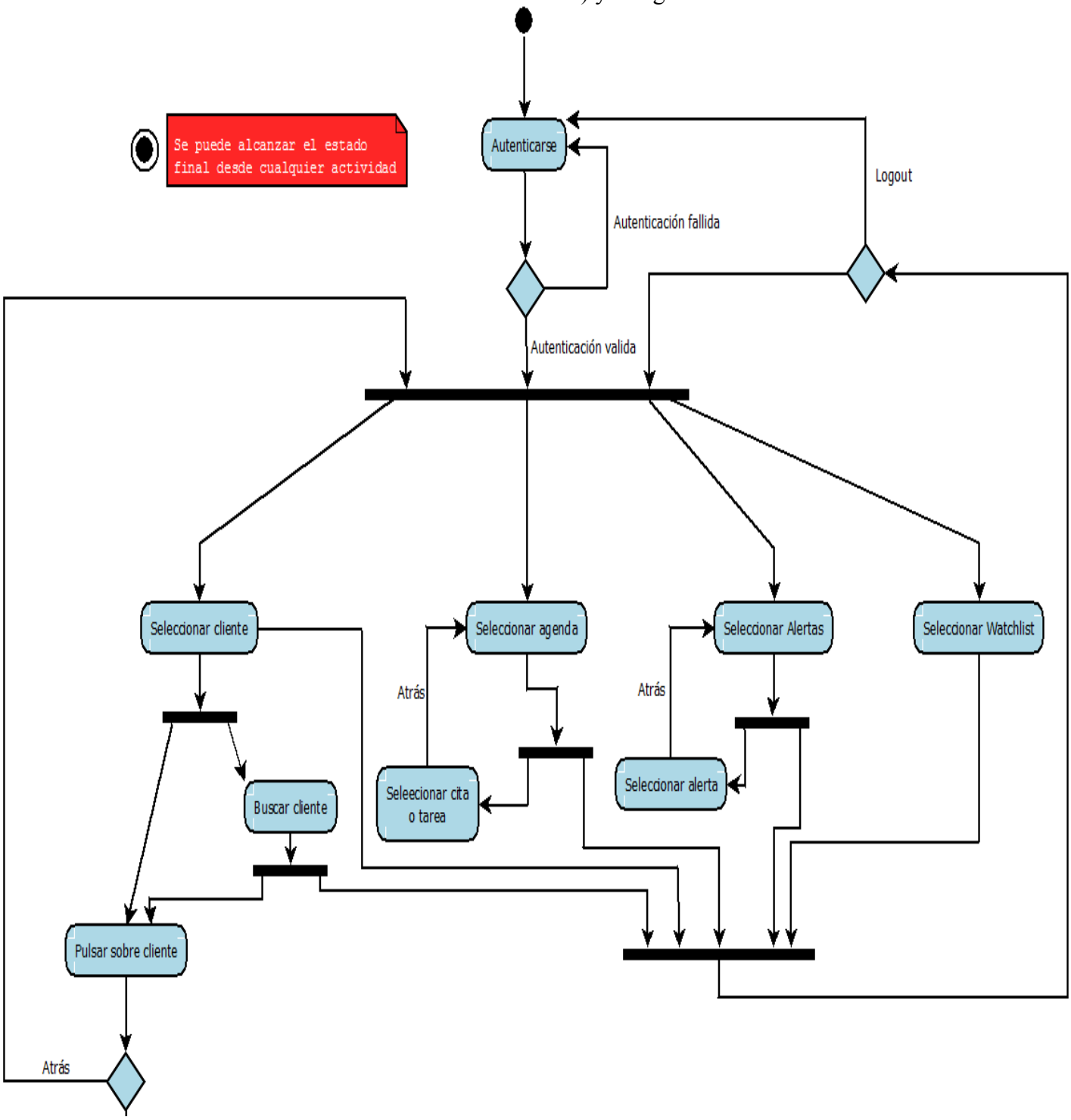


Figura 4: Diagrama de actividades (parte 1)

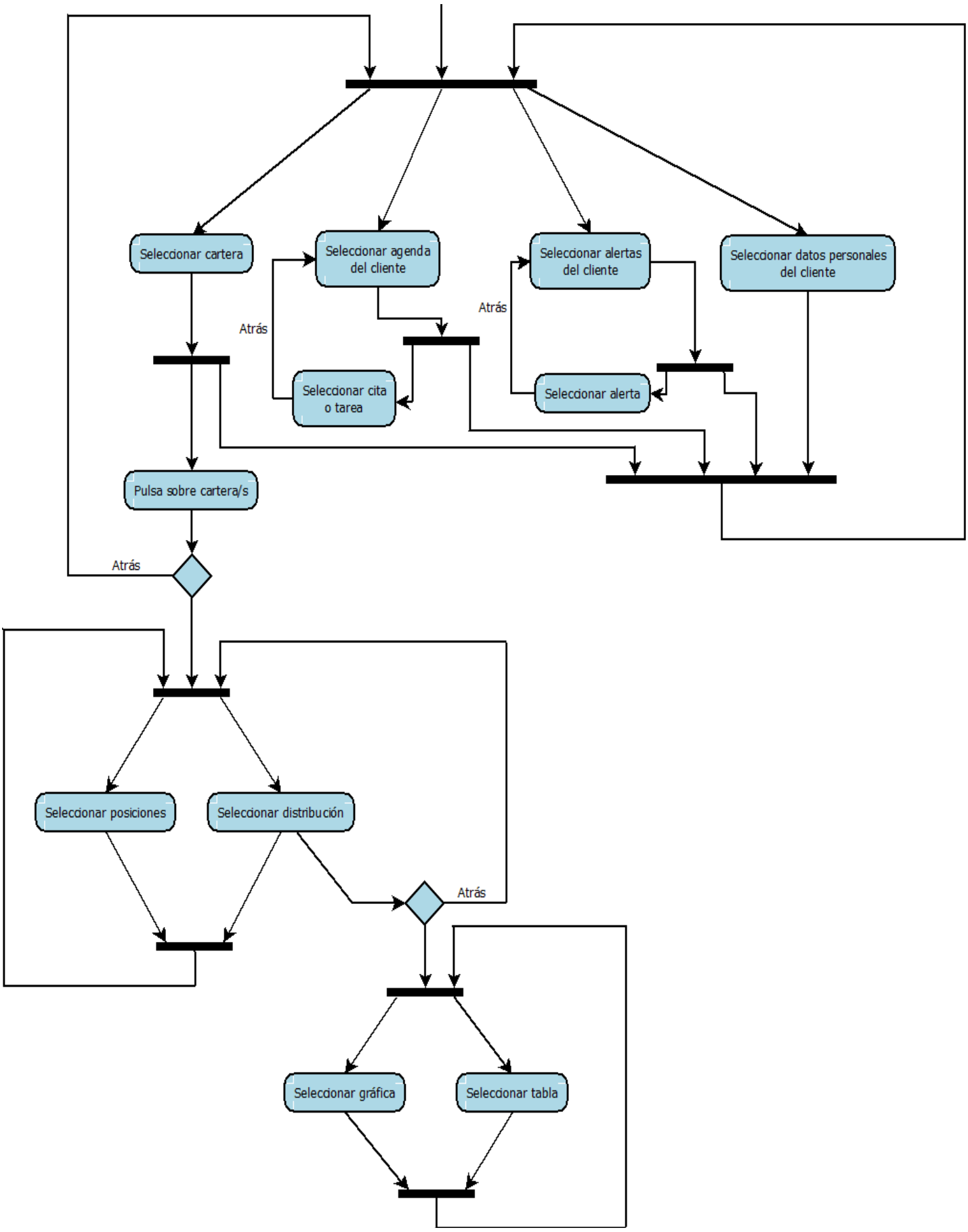


Figura 5: Diagrama de actividades (parte 2)

2.3.3 Diagrama de clases

En este apartado se muestra el diagrama de clases, el cual representa las clases que será necesario implementar para adecuarse a los requisitos especificados por la empresa. Debe comentarse que, en este caso, el diagrama no sólo se realiza en base a las especificaciones dadas por *Openfinance*, sino que se ha orientado también en torno a los servicios web con los que conectará la aplicación, puesto que deberá manejar la información que estos devuelven.

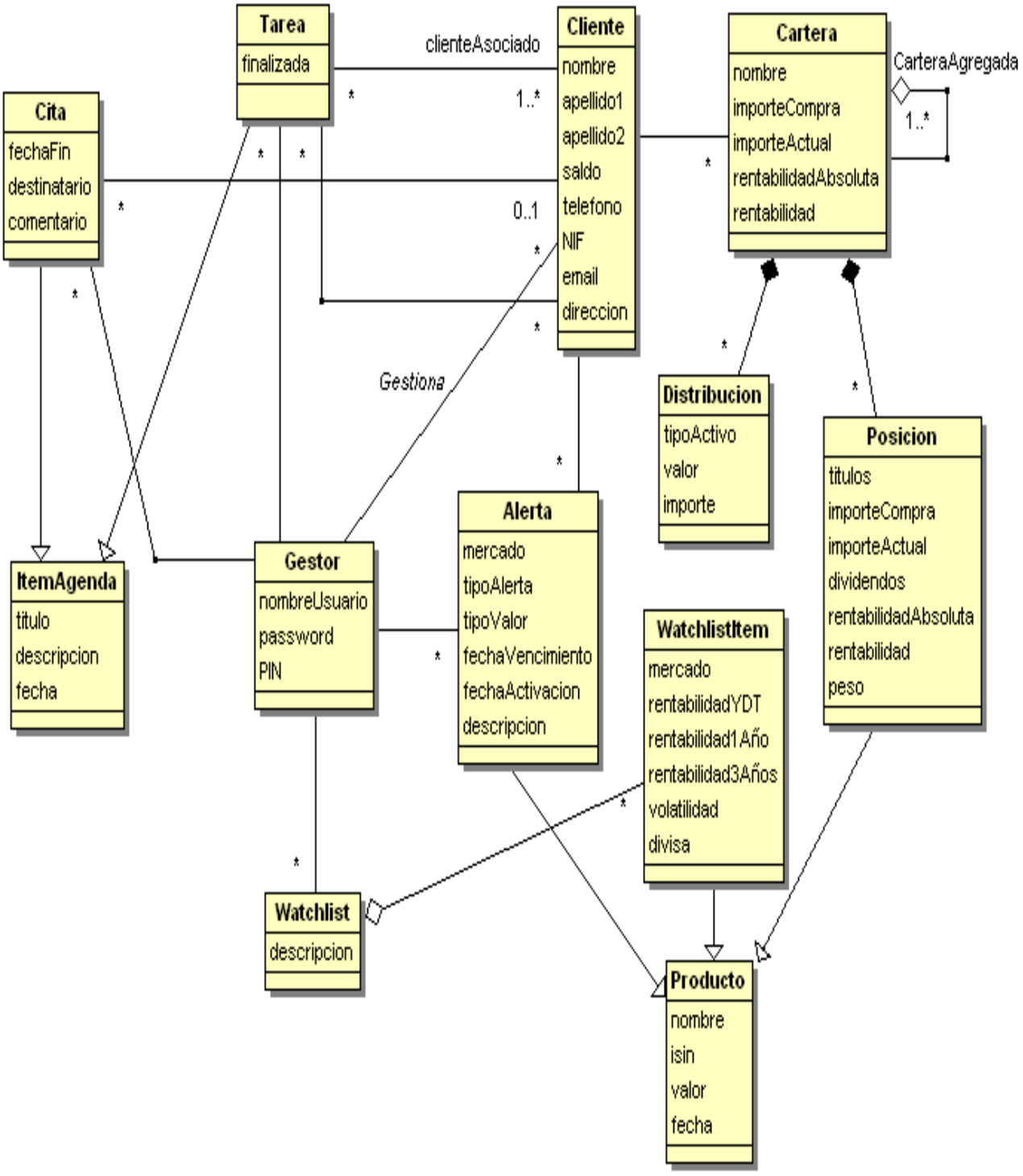


Figura 6: Diagrama de clases

3. Implementación

La implementación se ha llevado a cabo siguiendo un proceso iterativo horizontal de prototipos. A cada prototipo se añaden menús de la aplicación con sus correspondientes funcionalidades. El proceso seguido puede dividirse en cuatro prototipos, los cuales a su vez pueden desglosarse en una serie de pasos y funcionalidades.

- Primer prototipo: permite ver toda la información del usuario (gestor). Esto es el menú principal de la aplicación el cual se compone de cuatro pantallas distintas: listado de clientes del usuario, agenda del usuario, alertas del usuario y acceso al *Watchlist* (definición en apartado *Anexos*)del usuario.

- Segundo prototipo: en el listado de clientes el usuario puede seleccionar el que desee para acceder de esta forma al menú del cliente, el cual se compone de nuevo de cuatro pantallas: listado de las carteras de inversión del cliente, agenda del cliente, alertas del cliente e información completa del cliente. En esta última pantalla se ofrece una serie de posibilidades en función de la información de la que se disponga: si se dispone de los números de contacto se puede realizar una llamada telefónica desde esta ventana, o mandar un SMS si el número fuese de un teléfono móvil, así como mandar un email si se dispone de la dirección de correo electrónico del cliente.

- Tercer prototipo: en la pantalla del listado de carteras del cliente, el usuario puede seleccionar una o varias carteras sobre las que desee obtener más datos (posiciones y distribución). En caso de que no seleccione ninguna y acceda al siguiente menú la información a mostrar será de todas las carteras del listado. En este nuevo menú el usuario dispone de dos nuevas pantallas: la primera muestra un informe sobre las posiciones de la cartera o carteras seleccionadas para una determinada fecha; en la segunda ventana se muestra la distribución de las carteras, la cual puede verse en forma de gráfico o de tabla.

- Cuarto prototipo: una vez se han realizado todos los menús de la aplicación (menú del usuario, del cliente y de carteras), se incorporan las pantallas de presentación, configuración, autenticación y acceso a la aplicación mediante PIN. Esta última está deshabilitada por defecto, y su visibilidad se controla desde la ventana de configuración. Además de estas ventanas también se implementa el almacenamiento persistente de datos (datos de configuración, URLs, PIN...), así como mejoras en la interfaz y nuevas funcionalidades: posibilidad de que el usuario ordene los listados en base al atributo que elija, acceso a *GoogleMaps* desde la ventana de información del cliente si se dispone de su dirección o posibilidad de elegir el intervalo de fechas para el cual se quiere listar los datos correspondientes a la agenda.

Para llevar a cabo todo esto, el entorno de desarrollo escogido es *Eclipse*, debido a que en todos los libros de programación consultados y en la página oficial de *Android* es el recomendado y el único para el cual explican la instalación del *plugin* para usar el SDK. Los detalles de implementación respecto al SDK se verán más adelante.

4 Validación y pruebas

El SDK de *Android* permite integrar dispositivos virtuales para probar la aplicación en el entorno de desarrollo *Eclipse*. Queda a disposición del usuario crearse cuantos emuladores quiera y con las características que se desee. Esta flexibilidad permite entre otras cosas, establecer la versión de sistema operativo en la que se ejecutará, el tipo de pantalla, su resolución, cantidad de memoria...

Para realizar las pruebas se han creado dos emuladores: uno que implementa la versión 1.5 de *Android*, puesto que esta es la más común entre los terminales *Android* que se comercializan en España, y el segundo con la versión 2.1, puesto que es la última versión comercializada en España (ya existe la 2.2 pero no llegará a España hasta dentro de unos meses). Las principales diferencias entre ambos emuladores son la resolución de pantalla y la incorporación del botón de búsqueda (consultar anexos para una explicación de los botones *Android*) que en la versión 1.5 aún no existía.

En todo momento se realizaban pruebas en uno y otro emulador para poder observar las diferencias de ejecución entre una versión y otra. Las diferencias entre velocidades de ejecución y acceso a Internet eran casi siempre mínimas, siendo lo más destacable los cambios a nivel de interfaz debido a la leve diferencia de resolución de pantallas, permitiendo de esta manera ajustar mejor el tamaño y apariencia de la información a mostrar para una correcta presentación.

Como se comentaba en el apartado *3 Implementación*, la aplicación se ha ido desarrollando a través de un proceso iterativo de prototipos. Al término de cada uno de ellos se realizaba una pequeña fase de validación, dando la empresa en cada momento su visto bueno y opinión antes de continuar.

Esta validación era principalmente en cuanto a interfaz y cumplimiento de funcionalidades, puesto que las pruebas se realizaban sobre un emulador. De esta manera la optimización de código se deja para la fase final de la aplicación cuando ya pueda ser probada en un dispositivo real.

Una vez la aplicación se encontraba ya en fase de finalización se probó en un teléfono real, prestando principal atención a los tiempos de ejecución y la presentación de la información. La conclusión principal es que es necesario poner más ventanas de espera, puesto que en un dispositivo real, el usuario puede elegir entre emplear conexión a través de la red *3G*, con lo cual los tiempos son similares a los vistos en el emulador, o desconectarse de esta red para ahorro de batería, de esta manera la velocidad de conexión se reduce bastante y los retrasos en las transiciones de pantalla mientras se acceden a los datos son más notables.

Se toma también en consideración la forma con la que el sistema operativo *Android* gestiona el cambio de orientación de la pantalla, puesto que, al girar el dispositivo, lo que hace realmente es destruir la ventana en ejecución y volverla a crear. Esto hace necesario un tratamiento especial para evitar que vuelva a descargar de Internet los datos que ya tenemos, puesto que es lo que más ralentiza la aplicación.

Uno de los aspectos críticos a tener en cuenta para la optimización de la aplicación, es el tiempo de procesamiento de los datos obtenidos por los servicios web. Que como ya se ha comentado, devuelven información en ficheros *XML*.

En un principio, el tratamiento de éstos datos se realiza con las clases que proporciona la API de *Android*. No obstante, algunos de los manuales y foros consultados recomiendan usar la librería externa proporcionada por *DOM4J* para la lectura de este tipo de datos. Esto se debe a que esta librería hace uso de *XPATH*, lo cual permite la elaboración de un código más claro y fácil de mantener. Los detalles de implementación de cada una de las dos formas de procesamiento serán detallados en el apartado *5.5.2 Lectura de ficheros XML*.

No obstante, tras probar la aplicación y comprobar que con una carga considerable de datos el usuario ya tiene que esperar unos segundos hasta que éstos son procesados, se decide hacer una comparación del tiempo medio empleado por cada método de procesamiento.

Para realizar estas pruebas, se mide el tiempo que la aplicación tardar en descargar el fichero *XML* por Internet y procesarlo para representar los datos en la pantalla de la agenda del usuario. Se realiza en esta ventana, por que en ella se puede seleccionar para cuantos meses se desea mostrar la información de la agenda (por defecto muestra dos meses a partir del día actual). Cuando se selecciona mostrar para más de medio año, normalmente sobrepasa ya el segundo de retardo.

El proceso de comparación es el siguiente: se mide el tiempo que tarda en ejecutarse el método que se encarga de descargar y procesar el fichero *XML*. Esto se realiza con cada clase (API de *Android* o *DOM4J*) para un intervalo de dos meses, cuatro meses, ocho meses y catorce meses. La medición de cada intervalo se realiza cinco veces y se obtiene el tiempo medio, debido a que en el tiempo también influye el retardo de la conexión (aunque como es en un dispositivo móvil no hay más aplicaciones usando Internet al mismo tiempo).

El resultado de la comparación se muestra en la Figura 7. En ella se representa el tiempo tardado (en milisegundos) respecto al número de meses para los que se accede a la información. El lector, utilizando las clases proporcionadas por DOM4J, tarda casi siempre el doble que ejecutando el mismo proceso con las clases de la API de *Android*, llegando a ser la diferencia de dos segundos de espera para el usuario, cuando se muestran las entradas de la agenda para un año y dos meses.

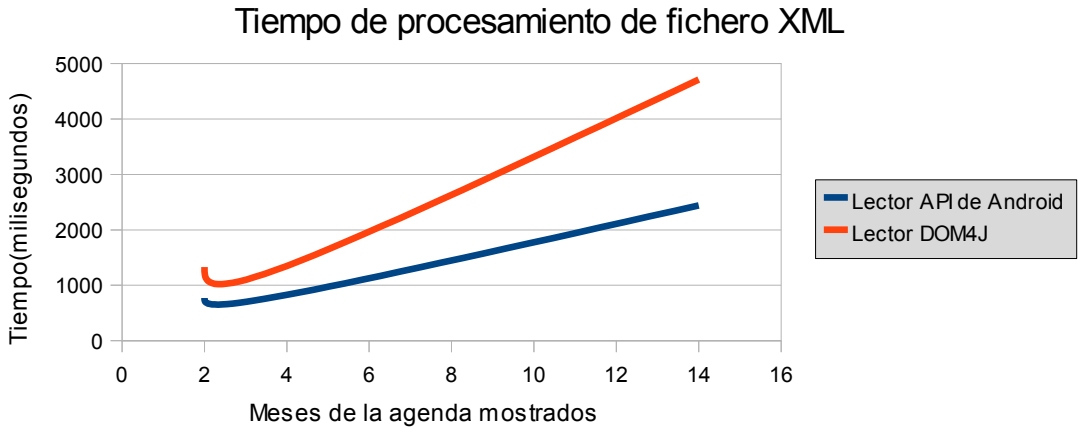


Figura 7: gráfica de tiempo procesamiento de un fichero XML

Lógicamente la diferencia de casi el doble de tiempo en todos los casos es un dato a tener muy en cuenta. Así que se toma la decisión de, una vez esté terminada la aplicación, retomar estas clases para la optimización máxima del código en cada uno de los casos.

Una vez se han modificado ambos métodos de lectura se vuelve a realizar las mediciones, obteniéndose el mismo resultado: dos segundos de diferencia con una gran cantidad de datos a procesar. Esta nueva comparativa se puede observar en la Figura 8. El propósito de esta gráfica no es compararla con la Figura 7, principalmente porque hay semanas de diferencia entre la recogida de datos para una y otra, y puesto que el usuario gestor con el que se prueba la aplicación es una cuenta de prueba para el *Workplace* común para toda la empresa, en estas semanas de diferencia se han introducido más clientes y entradas de la agenda del gestor de prueba. El propósito de esta gráfica es ver que tras optimizar todo lo posible cada una de las formas de procesamiento de datos, la *API* de *Android* sigue siendo en todos los casos la más eficiente.

Como se ha explicado, para un intervalo de dos meses, cuatro meses, ocho meses y catorce meses se realizan cinco mediciones con cada librería. En todos los casos la cota máxima de cada intervalo del *API* de *Android* es siempre inferior a la cota mínima de *DOM4J* para ese intervalo. Es decir, como conclusión no sólo sacamos que de media sea siempre más rápida la *API* de *Android*, sino que además, el caso peor de la *API* de *Android* para un mismo intervalo es siempre más rápido que el caso mejor de *DOM4J*.

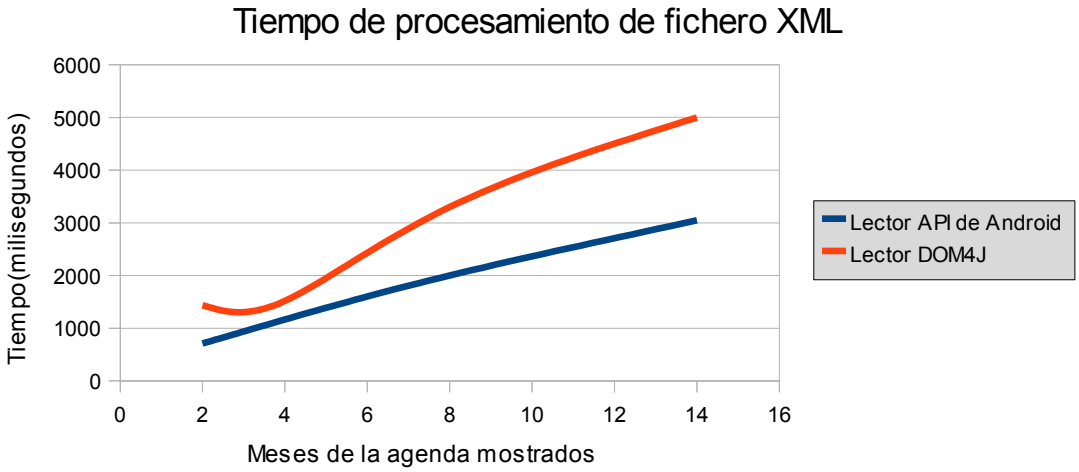


Figura 8: gráfica de tiempo procesamiento de un fichero XML con código optimizado

La principal recomendación para usar DOM4J es que permite realizar un código mucho más legible y fácil de mantener, pero lógicamente a la vista de los resultados la decisión final es la utilización de las clases proporcionadas por la API de Android, con lo cual se ha intentado dejar el código lo más reducido y fácil de mantener que se ha podido. La implementación de cada uno de ellos se detalla más adelante.

La conclusión final de estas pruebas realizadas es que, como ya se ha comentado, *Android* está orientado a hacer un uso lo más optimizado y eficiente de todos los recursos, con lo cual utilizar una librería externa a la API debería realizarse sólo en caso de que la tarea no se pueda realizar con las clases proporcionadas por ésta.

5 Programación con el *SDK de* Android

En este apartado se van a explicar las APIs y tecnologías utilizadas para la elaboración de este proyecto. *Android* es un sistema operativo para dispositivos móviles muy reciente, puesto que aún no se han cumplido dos años desde su primera aparición en el mercado; no obstante pese a su escaso tiempo de vida un programador con cierta experiencia se adapta enseguida a su programación.

Dado que es un tema bastante reciente, y para poner en situación al lector y permitirle conocer las características de este sistema operativo, a continuación se va a hacer una breve introducción sobre que es *Android* antes de explicar detalles de implementación.

5.1 Introducción

Los inicios de este sistema provienen del año 2005 cuando *Google* compra una pequeña compañía de nombre *Android Inc*, cuya finalidad es el desarrollo de aplicaciones para dispositivos móviles, centradas principalmente en plataformas *Web*.

La existencia del nuevo sistema operativo no se hace pública hasta el año 2007, momento en el que se anuncia la creación de la *Open Handset Aliance*. Esta organización es un consorcio de compañías relacionadas con las telecomunicaciones y la informática, cuyo principal objetivo es la promoción del sistema operativo *Android* y su política las licencias de código abierto.

La compañía *HTC* lanza a finales de septiembre de 2008 el primer teléfono móvil con este sistema operativo. Desde entonces, son ya varias las compañías que están apostando por *Android*. A modo de resumen estos son los tres nombres más importantes que se pueden relacionar con esta plataforma.

Google: esta compañía es en la actualidad una de las más poderosas del mundo, su rápido crecimiento en tan sólo unos pocos años de vida hizo que en 2007 se convirtiera en la marca más valorada del mundo por delante de *Microsoft* o *Coca-Cola,* empresas que acostumbraban a dominar esta posición.

Google lidera la mencionada *Open Handset Aliance.* El hecho de que esta compañía de tanta importancia sea la principal promotora de *Android* aporta dos grandes ventajas tanto a los usuarios de sus teléfonos móviles como a los desarrolladores de aplicaciones para esta plataforma.

La primera ventaja es que esta empresa apoya actualmente diversos proyectos de software libre y licencias de código abierto. Ocurre lo mismo con el caso de *Android*, lo que provoca que cualquier programador comience a trabajar cuando desee sin tener que pagar ninguna licencia para desarrollar aplicaciones o darlas a conocer en el *Android Market*. Este último es un repositorio de aplicaciones al que se puede acceder desde cualquier *Android*, quedando a elección del desarrollador que su aplicación sea gratuita o de pago.

La segunda ventaja, es la cantidad de productos pertenecientes a *Google* tales como *Gmail*, *Youtube*, *Google Maps*, *Google Calendar*... que cuentan actualmente con un nivel de aceptación muy alto por parte de los usuarios. Todos los *Android* cuentan con un acceso rápido y eficiente a estos productos y, al mismo tiempo, *Google* proporciona APIs para que el programador pueda incluir estos servicios en sus aplicaciones de forma muy sencilla.

Linux: *Android* es un sistema operativo abierto para dispositivos móviles, y se basa en la versión 2.6 del *Kernel de Linux* que actúa como una capa de abstracción entre el hardware y el resto del conjunto de software, además de prestar los servicios de seguridad, gestión de memoria, gestión de procesos... lo que hace que pueda ser adaptado con facilidad a otros dispositivos. Esto presenta una gran ventaja para las compañías fabricantes de teléfonos móviles.

En la Figura 9 podemos ver un diagrama que muestra los principales componentes del sistema operativo *Android*, los cuales se comentan a continuación.

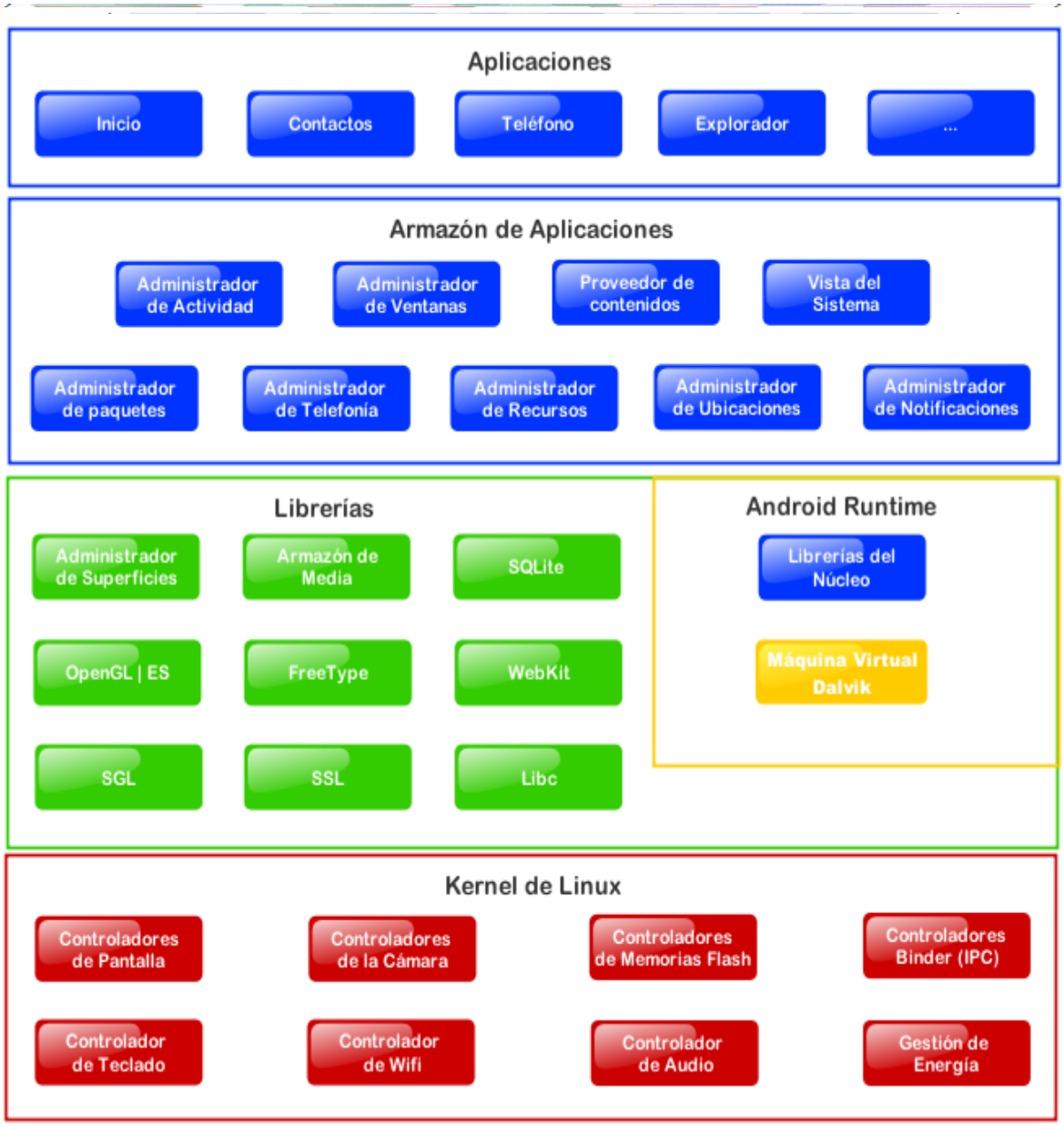


Figura 9: Arquitectura Android

- Aplicaciones: suministra un conjunto de aplicaciones que incluye cliente de correo electrónico, proveedor de SMS, calendario, mapas, navegador web, contactos... más todas aquellas que el usuario desee descargar del *Android Market*.

- *Framework* de aplicaciones: proporciona acceso a los marcos utilizado por la API de las aplicaciones básicas. La arquitectura de aplicaciones se ha diseñado para simplificar la reutilización de componentes. De esta forma cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación podrá hacer uso de esas capacidades (sujeto a las limitaciones de seguridad impuestas por el marco). Esto hecho permite la reutilización de componentes por parte del usuario.

  Detrás de todas las aplicaciones existen un conjunto de servicios y sistemas que incluyen un conjunto de vistas para construir las pantallas, así como proveedores de contenido que permiten que las aplicaciones de acceso a los datos de otras aplicaciones (como Contactos), o para compartir sus propios datos.

- Librerías: incluye un conjunto de librerías escritas en C y C++ utilizado diversos componentes del sistema . Estas capacidades están expuestas a los desarrolladores a través del *Framework*. El conjunto de las mismas puede observarse en la Figura.

- *Android Runtime*: incluye un conjunto de librerías que proporciona la mayor parte de la funcionalidad disponible en el núcleo de las bibliotecas y esta basado en el lenguaje de programación *Java*. Cada aplicación se ejecuta como un proceso independiente, con su propia instancia de la máquina virtual *Dalvík* que se ha escrito de modo que un dispositivo puede ejecutar múltiples máquinas virtuales de manera eficiente. La máquina virtual de *Dalvík* ejecuta archivos *.dex* cuyo formato está optimizado para un consumo de memoria mínima. La máquina virtual se basa en registros, y corre clases compiladas por un compilador de lenguaje Java que se han transformado con la herramienta incluida *dx*. La máquina virtual de *Dalvík* se basa en el K*ernel de Linux* para la funcionalidad subyacente como subprocesos y de gestión de memoria a bajo nivel.

- *Kernel de Linux*: explicado anteriormente antes de mostrar el diagrama.

Java: el tercer nombre importante asociado al sistema operativo *Android* es *Java*. Como se comentaba anteriormente, a pesar del poco tiempo de vida de esta plataforma cualquier programador se adapta a su funcionamiento en muy poco tiempo, dado que el código fuente de una aplicación se escribe en *Java* y este es uno de los lenguajes de programación más utilizados en la actualidad. Más adelante se explicarán detalles de programación.

Además, como ventaja añadida el "principal inconveniente" que se suele asociar con *Java* es su velocidad de ejecución. En este caso, como se ha explicado antes, ese problema ni siquiera existe dado que, aunque la aplicación se escribe en *Java*, no es una aplicación *Java* ya que no se ejecuta en una máquina virtual de *Java*. De esta forma el programador cuenta con todas las ventajas de este lenguaje, y desaparece una de sus mayores desventajas.

5.2 Especificaciones técnicas

Antes de comenzar con los detalles de programación, se explica todo lo necesario para comenzar a desarrollar aplicaciones así como los pasos necesarios para la instalación de todos sus componentes. Estos pasos pueden seguirse también de forma resumida en la página oficial de *Android*, en el apartado *Developers* → *Download SDK* desde el cual, además, podemos descargar todos los elementos necesarios que a continuación se explican.

JDK (Java Development Kit): en primer lugar es necesario tener instalado Java, la versión mínima del JDK debe ser la 5. Puede comprobarse si se dispone de la versión necesaria introduciendo en la ventana de comandos el siguiente comando (debería mostrar al menos 1.5 para que sea la correcta):

```
javac -version
```

Android SDK: mientras que el JDK nos proporciona lo necesario para utilizar las clases de Java, el SDK nos dará acceso a las clases necesarias para construir una aplicación *Android*. Para ello lo descargamos de la página oficial de *Android* en el apartado comentado anteriormente, seleccionando la versión correspondiente a la máquina en la que se vaya a trabajar (*Mac*, *Windows* o *Linux*). Esto nos permite guardar un archivo comprimido, el cual únicamente hay que descomprimir y guardar la carpeta resultante en el directorio en el que queramos tener el SDK.

Tanto la página oficial como los libros consultados dicen que no es necesario nada más y que simplemente debe descomprimirse el archivo y posicionarlo donde deseemos. No obstante seguramente la carpeta *add-ons* se encuentre vacía, lo que provocará que, aunque realizemos bien todos los pasos no podamos crear un proyecto en *Eclipse*; si fuera así deberá ejecutarse el archivo "SDK Setup" y descargar desde allí todas las versiones del SDK. Si hubiera algún problema con la descarga, debe irse a la ventana de *Settings* y activar "Force https://..." como muestra la Figura 10.

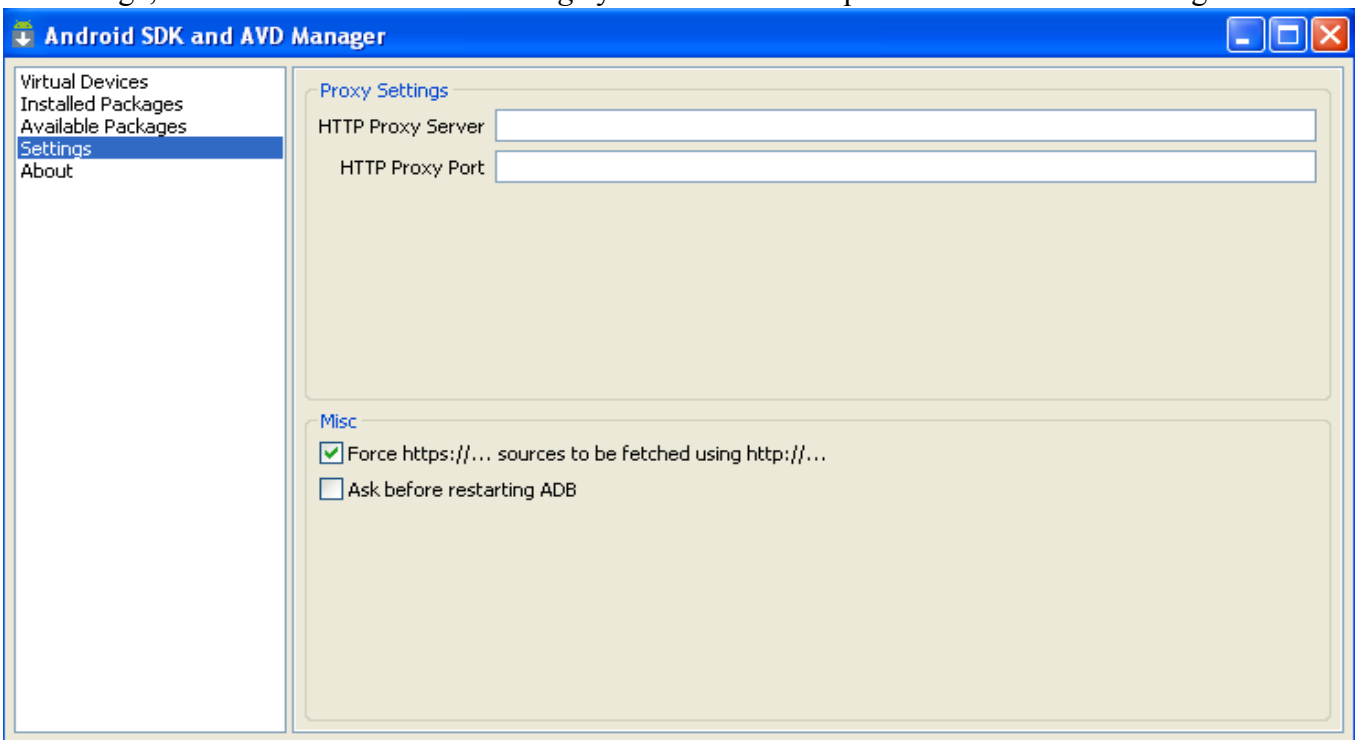


Figura 10: Ventana settings del AVD Manager

Una vez tenemos datos en la carpeta *add-ons,* lo siguiente es crear un dispositivo virtual (puede realizarse en cualquier momento antes de que vayamos a probar la aplicación), lo que nos permitirá probar la aplicación en el emulador antes de instalarla en un dispositivo real.

Se pueden crear cuantos emuladores se quiera y con diversas características. Por ejemplo se puede crear uno con la versión 1.5 de *Android* que fue la primera en comercializarse en España y otro con la última versión para comprobar que funcione todo correctamente en diferentes versiones. Para crearlos, en la ventana que muestra la Figura 10 tenemos que ir a la primera opción: *Virtual Devices* y pulsar "New". En la nueva ventana deberemos establecer un nombre identificador para cada dispositivo virtual, en el apartado *Target* la versión de sistema operativo que deseemos y el tamaño que queremos que tenga su tarjeta de memoria.

Eclipse IDE: es un entorno de desarrollo para compilar y ejecutar aplicaciones. Si se prefiere *NetBeans* también cuenta con su *plugin* para programar en *Android*, pero todos los libros y la página oficial de *Android* se centran en la herramienta *Eclipse.*.

La versión mínima necesaria es la 3.4, ya sea del *Classic* o para desarrolladores de *Java*. En este caso sí que es suficiente con sólo descargar el archivo y descomprimirlo, ya que no es necesaria ninguna instalación.

Una vez realizados todos los pasos anteriores sólo queda configurar Eclipse correctamente para poder implementar aplicaciones con el SDK de *Android*. Los pasos dados se han realizado para el *Eclipse Classic 3.5.2* con lo cual dependiendo de la versión puede que los nombres de las pestañas o su localización varíen ligeramente. Una vez hemos abierto *Eclips*e hay que realizar las siguientes acciones:

- *Help → Install New Software.*
- En la nueva ventana pulsar *Add.*
- En la ventana emergente poner en el campo *Name* el nombre que queramos darle al *plugin* y en *Location* copiar este enlace y pulsar *OK*:

  https://dl-ssl.google.com/android/eclipse/
- Cuando finalice la descarga seleccionar todos los elementos que aparezcan, pulsar *Next*, aceptar todos los acuerdos de licencia y finalizar, tras lo cual será necesario reiniciar Eclipse.

Una vez se ha instalado el *plugin* el último paso es configurarlo. Parra ello hay que seguir los siguientes pasos:

- *Window → Preferences.*
- En el panel seleccionar *Android* y en *SDK Location* escribir la ruta en la que hayamos guardado la carpeta mencionada anteriormente. A continuación, pulsar *Apply* y *OK.*

En caso de que tras pulsar *Apply* la lista del panel continúe vacía deberá realizarse el paso comentado anteriormente sobre el archivo "SDK Setup", o acceder a dicha ventana desde *Eclipse* pulsando en el nuevo icono de *Android* que aparece en la barra de tareas de la aplicación.

5.3 Aplicación Android

A continuación se explicará las clases básicas que proporciona el SDK, así como las principales funcionalidades implementadas para la elaboración de este proyecto. Primero se comentará la estructura de carpetas de un proyecto *Android,* a continuación se ofrecerán comentarios sobre la API y código, y por último sobre la construcción de interfaces gráficas, las cuales pueden especificarse mediante código o a través de archivos *XML*.

5.3.1 Estructura de carpetas

En la Figura 11 podemos ver como se organiza el sistema de directorios de un proyecto *Android*. Estos son los tres directorios más importantes:

- *SRC*: en esta carpeta se incluyen las clases con el código fuente de la aplicación al igual que en cualquier otro proyecto *Java*. Dentro de este directorio lo podemos organizar como queramos, y con cuantas subcarpetas como sean necesarias, con la peculiaridad de que, en un proyecto *Android,* las carpetas contenidas en *SRC* deben tener como mínimo dos niveles de profundidad. En este caso dentro de la carpeta *SRC* está la carpeta *openfinance*, que contiene la carpeta *workplace* y en esa ya podemos crear cuantas queramos.

- *Gen*: esta carpeta es generada y controlada automáticamente por el entorno de desarrollo, no debe ser modificada puesto que se actualiza sola cada vez que se haga algún cambio dentro de la carpeta *res*. Sirve, por lo tanto, como interfaz entre la carpeta *res* y el código fuente contenido en *src*. Con lo cual, cada vez que quiera accederse mediante código a un elemento de la carpeta *res* (que será explicada a continuación) debe hacerse a través de la clase *R.java*. La siguiente línea de código muestra un ejemplo:

```
.setIndicator("Clientes", getResources().getDrawable(R.drawable.clientes))
```

  en ella se pone en la pestaña "Clientes" como icono la imagen que hemos puesto con el nombre *clientes* en la carpeta *drawable*

- *Res*: en esta carpeta se incluye todo lo que no sea código *Java*, pero que queramos que aparezca en la aplicación. Por defecto están creadas las carpetas *drawable* (para colocar imágenes), *layout* (para crear archivos *XML* que definen interfaces) y *values* (para colocar *Strings* y *arrays*). Pero, como se observa en la Figura 11, podemos añadir más. En este caso tenemos la carpeta *anim* en la que dentro hay un fichero *XML* que define una animación que se utiliza cuando un dato es introducido incorrectamente.

  Cada vez que introduzcamos un elemento nuevo en una de estas carpetas, o modifiquemos el contenido de algo que ya hubiera, le asignará un identificador en la carpeta *R.java* con el nombre que tenga el archivo si todo es correcto. En el caso de los archivos de la carpeta *layout* el nombre debe estar escrito en caracteres en minúsculas.

Por último observamos la existencia del archivo *AndroidManifest.xml*, el cual tiene un papel muy importante para el correcto funcionamiento de la aplicación.

En él deben definirse todos los permisos de los que haga uso la aplicación, puesto que serán mostrados al usuario en el momento de instalación para que decida si quiere instalarla o no. Si, por ejemplo, en algún momento enviamos un SMS pero no definimos en este archivo el permiso para mandarlos, la aplicación lanzará una excepción y finalizará cuando se fuese a producir esta acción. Asi siendo, para que esto no pase, escribimos estos en el archivo de manifiesto:

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

También debe especificar el nombre de todas las actividades (el concepto de actividad será explicado a continuación) que aparecen en la aplicación, sino al igual que en el caso anterior saltará una excepción cuando la actividad correspondiente vaya a ser realizada. También pueden definirse ciertos aspectos de la actividad como su título y su apariencia como muestra la siguiente línea de este fichero:

```
<activity android:name=".Enviar" android:label="@string/app_name"
android:theme="@android:style/Theme.Dialog" />
```

En esta línea se especifica que una de las actividades de la aplicación se llama *Enviar.java* dentro de la carpeta *SRC*, que su título es el contenido por el *String* de nombre *app_name* de la carpeta *values/string.xml* y que su apariencia será de la de una ventana de diálogo. Si este último atributo no se hubiera especificado, el tamaño de la actividad cuando se mostrase ocuparía toda la pantalla.

Por último debemos especificar también que actividad será la primera a mostrarse en la aplicación. Esto se hace de la siguiente manera:

```
<activity android:name=".Presentacion" android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

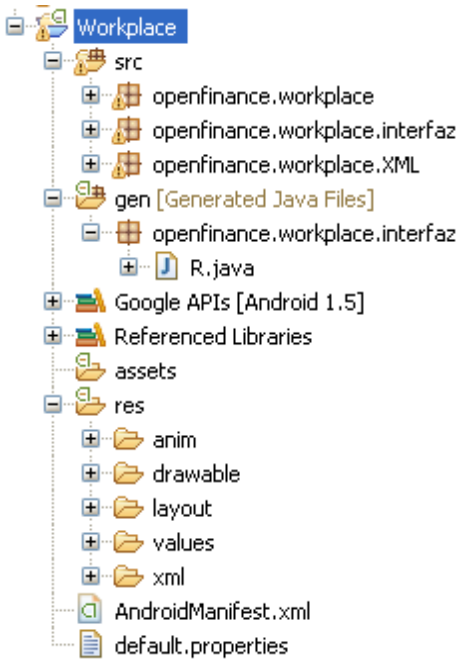


Figura 11: estructura de carpetas de un proyecto Android

5.3.2 API de Android

En este apartado se comentan las clases que se han utilizado pertenecientes al API de *Android*. Se utilizan como si fueran cualquier otra clase de *Java*.

5.3.2.1 Activity

Es la principal clase de *Android*, y cada pantalla que se muestra en la aplicación es una clase que hereda de *Activity.* Así siendo, podríamos decir que una actividad equivale a una pantalla. En la Figura 12 se muestra el ciclo de vida de una actividad.

En el diagrama se observan los métodos que podemos sobrescribir si queremos realizar una acción en concreto, y cuando se ejecutan. No es necesario ni obligatorio implementar ninguno de ellos; el único que debemos implementar siempre (al menos si queremos que la actividad muestre algo) es *onCreate()* donde definimos todo lo queramos hacer antes de que la pantalla se muestre.

Por ejemplo en el caso de que a una actividad se pueda regresar desde diversas actividades, y queremos hacer cosas distintas dependiendo de cuál vengamos, la forma idónea no sería con *onResume()* puesto que, si queremos, en el momento de llamar desde una actividad a otra podemos dejarla esperando el resultado de la ejecución con un método más eficiente, que será detallado cuando se explique la clase *Intent*.

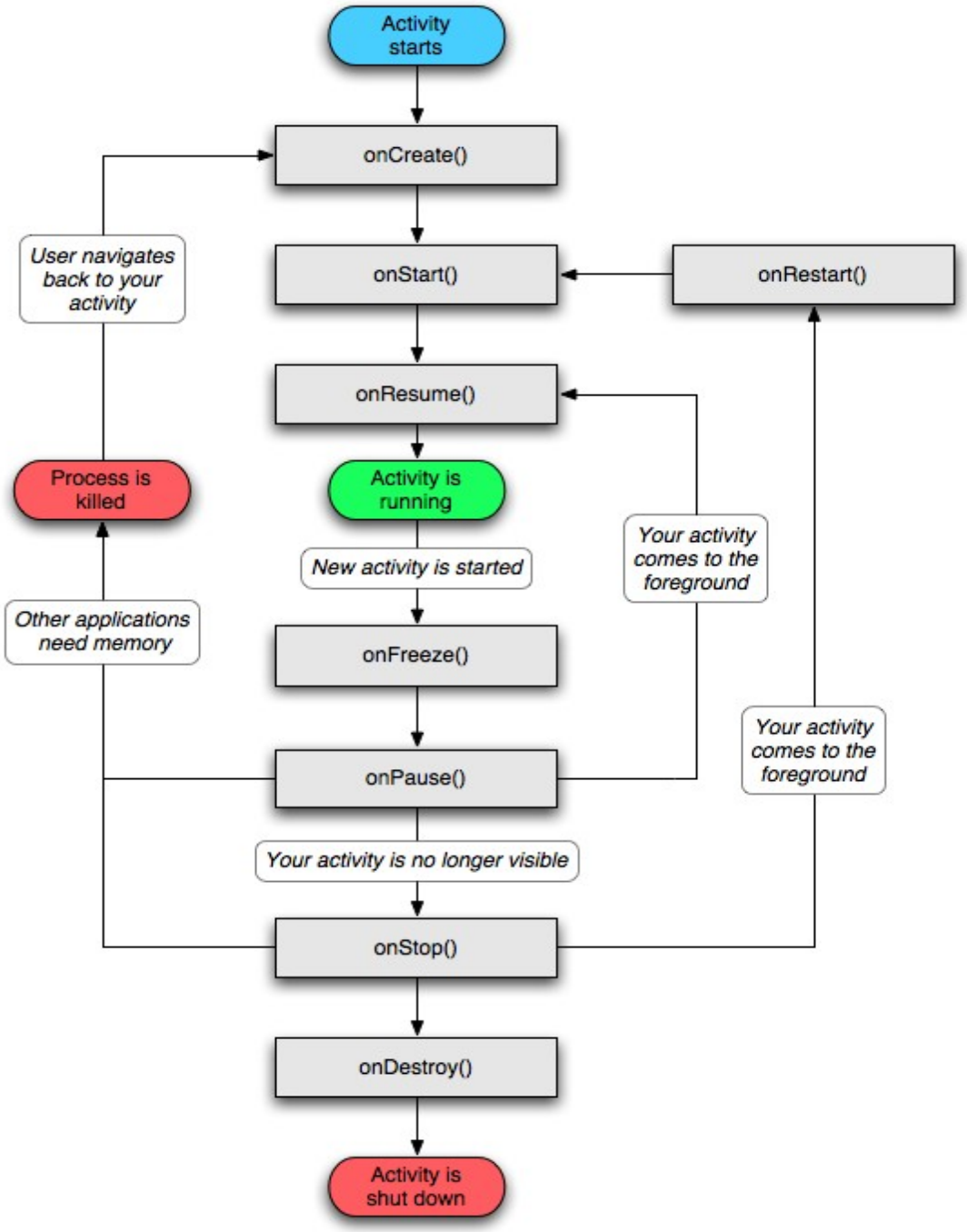


Figura 12: Ciclo de vida de Activity

Como curiosidad, y cosa muy importante a tener en cuenta en el caso de esta aplicación ya que casi todas sus actividades leen datos de Internet, hay que comentar que no hay método definido en el ciclo de vida para detectar cuando se cambia la orientación de la pantalla. Si vamos a leer datos de Internet, lo normal es hacerlo dentro del *onCreate()* puesto que sólo se ejecuta una vez y si pasamos a otra actividad al volver el *onCreate()* no se ejecutaría, con lo cual no se vuelven a leer los datos. Sin embargo, cuando se cambia la orientación, la actividad es destruida y se vuelve a ejecutar el *onCreate()*. Es decir, si fueran datos muy costosos de leer por Internet, y si por descuido o por voluntad propia el usuario mueve el dispositivo y la pantalla gira, deberían volverse a cargar todos los datos.

Afortunadamente para este caso, el SDK de *Android* permite almacenar un objeto cuando la pantalla gira y recuperarlo cuando deseemos. Esto se hace de la siguiente manera:

```
//SE EJECUTA CUANDO SE CAMBIA LA ORIENTACIÓN DE LA PANTALLA
public Object onRetainNonConfigurationInstance() {
    return listaClientes;
}
```

En este caso, cuando el usuario cambie la orientación de la pantalla, almacenará automáticamente un *ArrayList* que contiene todos los clientes del gestor. Este objeto se puede recuperar en cualquier momento con el método `getLastNonConfigurationInstance()`, el cual devolverá el *Object* que hubiéramos guardado, o *null* si ese método no se ejecutó. De esta manera, con el siguiente código en el *onCreate()*, evitamos que se lean de Internet todos los clientes cada vez que se gire el dispositivo.

```
final Object datos = getLastNonConfigurationInstance();
//SI DATOS != NULL ES POR QUE SE CAMBIÓ ORIENTACIÓN Y YA TENEMOS LOS
//DATOS LEIDOS
if( datos == null ) {
  listaClientes = getClientes();
} else {
  listaClientes = (ArrayList<Cliente>) datos;
}
```

Como se ha dicho al principio de este apartado, una actividad es una pantalla, con lo cual podemos hacer que aparezcan elementos visuales en ella. Además, cada uno de ellos debe ser un objeto que herede de la clase *View,* tal y como se explicará más adelante. También se ha comentado que estos elementos visuales pueden ser definidos mediante código o a través de ficheros XML. Para realizarlo de la segunda forma se hace mediante esta línea de código:

```
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.listado_carteras_cliente);
      //...
}
```

En este caso, este es el comienzo del *onCreate()* en el que se mostrará la tabla con las carteras de inversión de un determinado cliente. Antes de hacer nada con ningún elemento visual, debe llamarse al método de la clase superior. Como en este caso queremos que la apariencia visual sea la que tenemos definida en un fichero *XML*, llamamos al método que toma un contenido visual y a través de clase *R.java* explicada anteriormente accedemos al *XML* que queremos mostrar.

La clase que queramos mostrar también puede heredar de otra clase que herede de *Activity,* como por ejemplo *TabActivity* que construirá la pantalla con un panel con tantas pestañas como definamos, un *ListActivity* que es para mostrar elementos en forma de lista y sus correspondientes métodos para tratarla, o por ejemplo *MapActivity* proporcionada por un API de *Google*. También podemos reutilizar las que ya hemos creado nosotros.

5.3.2.2 Intent

La mejor forma de definir la clase *Intent* es como un nexo de unión. Su uso más frecuente es para pasar de un *Activity* a otro, o para realizar una acción en una actividad.

Esta clase permite que el programador se despreocupe del flujo de datos de su aplicación, ya que la clase *Intent* se encarga de ello y puesto que todos los *Android* disponen de un botón "volver" no es necesario ningún control de flujo.

Si por ejemplo tenemos una actividad a la que se puede llegar desde cuatro actividades distintas, la clase *Intent* se encarga de controlar a cual debe regresar cuándo se pulse el botón del dispositivo para volver. Obviamente si es necesario o se desea, a través de todos los métodos que definen el ciclo de vida se podría realizar un control de flujo de datos.

En esta línea de código se muestra como pasar de la actividad que lista los clientes a la actividad que muestra el menú del cliente que se ha seleccionado. Simplemente debe construirse un objeto de clase *Intent* cuyos parámetros sean donde estamos y donde queremos ir:

```
Intent intent = new Intent(ListaClientes.this, MenuCliente.class);
startActivity(intent);
```

Como se comentaba al explicar la clase *Activity*, en este ejemplo pudiera ser que al regresar de *MenuCliente* queramos hacer algo en *ListaClientes* y se podría realizar esta acción si implementamos *onResume()* ya que al volver de otra actividad es el método que se ejecuta. Puede darse el caso que desde una actividad podamos ir a muchas otras, pero que no deseemos que se realice la misma acción siempre que regresemos a *ListaClientes.* En este caso hay una alternativa a poner el código en *onResume(),* ya que existe otro método para lanzar el *Intent* que dejará la actividad que ejecuta el *Intent* en espera de el resultado de la actividad lanzada. Este código es un ejemplo de ello:

```
Intent intentFecha = new Intent(ListaAgenda.this, IntervaloFechas.class);
startActivityForResult(intentFecha, FIN_CAMBIO_FECHA);
```

En este ejemplo, se abre la ventana para seleccionar el intervalo de fechas para el cual el usuario quiere que se muestre su agenda. De esta manera, al regresar de *ListaAgenda* puede que haya ocurrido una de estas tres cosas: el usuario ha cambiado el intervalo de fechas, no lo ha cambiado o ha cerrado la ventana pulsando el botón "volver" del dispositivo. Lógicamente, sólo queremos realizar un cambio en el primero de los tres casos.

Para ello, con *startActivityForResult()* quedamos a la espera de que la nueva actividad finalice y le damos un identificador. En este caso será el valor que tenga el atributo *FIN_CAMBIO_FECHA*. En la clase *IntervaloFechas* podemos asociar un resultado de ejecución correcto con las siguientes líneas de código:

```
setResult(RESULT_OK);
finish();
```

Cuando queramos cerrar una clase *Activity* lo podemos hacer con la llamada al método *finish()*. Antes de hacerlo, si queremos, le ponemos el resultado de la acción de la actividad. En el caso que estamos tratando, si el usuario pulsa el botón para guardar el intervalo de fechas, pero no las ha cambiado no haremos el *setResult(RESULT_OK)*; si las ha modificado sí lo haremos. *RESULT_OK* en este caso no es una constante creada por nosotros como *FIN_CAMBIO_FECHA*, sino que pertenece a la clase *Activity.*

Una vez que la actividad *IntervaloFechas* termine se ejecutará el siguiente método en la clase *ListaAgenda*:

```
protected void onActivityResult(int requestCode, int resultCode, Intent
      data) {
      super.onActivityResult(requestCode, resultCode, data);
      if( requestCode == FIN_CAMBIO_FECHA && resultCode == RESULT_OK ) {
            //...
      }
}
```

Este método nos devuelve de qué actividad volvemos y cual ha sido su resultado; si la actividad termina pulsando el botón "volver" el resultado sería *RESULT_CANCELED*. De esta forma se controla de manera más sencilla e intuitiva las acciones que queremos hacer al regresar a una pantalla que con la clase *onResume()*.

Como se mencionaba al principio, *Intent* podría considerarse un nexo de unión o para realizar acciones, con lo cual no sirve exclusivamente para pasar de una actividad a otra, sino que, como se muestra en este ejemplo, también sirve para lanzar acciones que el dispositivo móvil lleva integradas, como por ejemplo realizar una llamada. En este caso, al crear el *Intent* debemos pasarle el número al que se desea llamar:

```
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:" +
      numeroTel));
startActivity(intent);
```

5.3.2.3 View

Todo objeto que queremos que se muestre por pantalla debe heredar de la clase *View*. Podemos definir nuestras propias clases que hereden de *View,* o usar las que nos proporciona el API de *Android*. Los objetos *View* los podemos organizar y ordenar dentro de objetos de clase *Layout*.

Con un *Layout*, por ejemplo, podríamos definir que los componentes se agregaran de manera horizontal o vertical en una línea. También definirlos en un formato de tabla controlando las posiciones en columnas y filas. Entre los *layouts* proporcionados por *Android* los siguientes son los más básicos:

- *LinearLayout*: este *Layout* ordena a sus elementos en una línea en sentido horizontal o vertical, agregándolos uno por uno en el orden en que se van definiendo.

- *FrameLayout*: es el tipo más simple de *Layout* que permite agregar un solo elemento en un espacio en blanco definido.

- *TableLayout*: este *Layout* agrega sus elementos en columnas y filas. Las filas deben ser insertadas una por una con un *RowLayout*.

- *RelativeLayout*: permite agregar elementos respecto a uno previo. Esto significa que el posicionamiento se administra según la posición del elemento anterior.

Las interfaces se definen de forma jerárquica, con lo cual podemos usar tantos elementos de los descritos anteriormente como queramos, combinándolos o introduciendo unos dentro de otros. Otra forma de organizar los objetos de clase *View* es utilizar los *ViewGroups*, los cuales también pueden contener *Layouts* si se desea. Estos son los más comunes:

- *Gallery*: utilizado para desplegar listados de imágenes en un componente con Scroll.

- *GridView*: despliega una tabla con *Scroll* de m columnas y n filas.

- *ListView*: despliega una lista con *Scroll* de una columna.

- *ScrollView*: una columna vertical de elementos con *Scroll*. Dentro de la jerarquía, el *ScrollView* sólo puede contener un elemento, si queremos que el *scroll* se realice sobre varios *Views* estos deberán estar dentro de un *ViewGroup* o *Layout* el cual será el que este dentro del *ScrollView*.

Una vez sabemos organizar los objetos de clase View ya podemos añadir los que deseemos. La elaboración de la interfaz será explicada más adelante; ahora simplemente se hará una descripción de los objetos de clase View más utilizados:

- *Button*: elemento que pinta un botón en la pantalla con el objetivo de que el usuario realice alguna interacción con él. Se le puede colocar una imagen de fondo, pero para construir un botón que sea una imagen existe también una clase similar llamada *ImageButton*.

- *CheckBox*: caja de texto seleccionable que permite evaluar dos estados del elemento: seleccionado y no seleccionado.

- *EditText*: elemento que permite editar texto dentro.

- *TextView*: etiqueta de texto no editable por el usuario.

- *RadioButton*: elemento seleccionable parecido al *CheckBox*, pero con la diferencia de que una vez que se ha seleccionado no regresa a su estado anterior si este está definido dentro de un RadioGroup.

- *Spinner*: listado de elementos oculto que solo aparece cuando se pulsa sobre él. Cuando está oculto sólo se puede ver el elemento seleccionado de todo el listado.

Con estas tres clases, y sus correspondientes clases descendientes, se podrían construir aplicaciones bastante complejas. Podemos resumirlas de la siguiente manera: una *Activity* es una pantalla a mostrar, todo lo queramos que aparezca en la pantalla tiene que ser descendiente de *View* o agrupaciones de estos objetos como *Layouts* y *ViewGroups*. Por último, los *Intent* nos permiten pasar de un *Activity* a otro, o realizar acciones definidas por el teléfono sobre él.

También existe la clase *Service*, de funciones similares a un *Activity* pero ejecutándose en segundo plano. Es decir, es como una actividad en *background* que podemos estar ejecutando y que no mostrará nada por pantalla durante su ejecución.

A continuación, en vez de explicar clases concretas se explicarán acciones que se han implementado para esta aplicación, describiendo en cada una de ellas las clases del API necesarias para llevarlo a cabo.

5.3.2.4 Leer información de Internet

Como se comentó con anterioridad, la forma de conectar con el servidor es mediante un servicio web *REST* y autenticación *Basic*. A continuación se va a explicar únicamente como hacer una petición *REST* desde *Android* para la obtención de los datos. La conversión del usuario y contraseña a *Base64* y el procesamiento de un fichero *XML* se han realizado con librerías que no pertenecen a la API, con lo cual se explicará en el siguiente apartado.

Antes de detallar los aspectos de programación, se explica en que consiste la petición *REST* mediante autenticación *Basic*. Este proceso puede verse en la Figura 13, y es comentado a continuación.

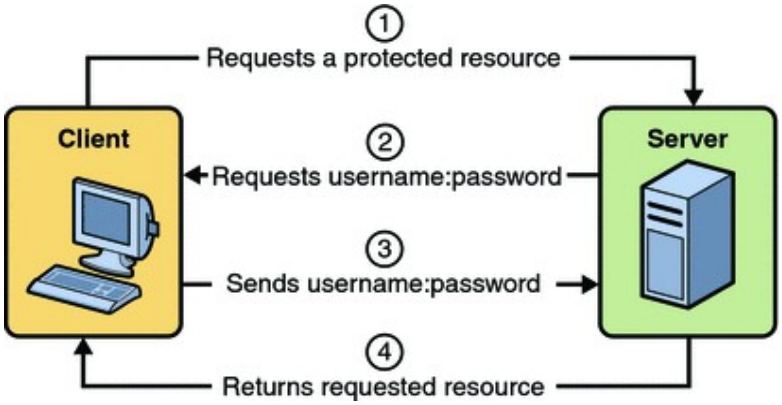


Figura 13: Autenticación Basic

1. Petición de un recurso protegido: el cliente solicita una *URL* que requiere autenticación y el cliente no ha proporcionado todavía ninguna información, únicamente ha dicho que quiere acceder a ese recurso.

2. Respuesta del servidor con petición de usuario y contraseña: el servidor responde con un error 401 y comienza el proceso de autenticación. El cliente recibe la cabecera de error 401 y muestra un diálogo al usuario pidiéndole su nombre de usuario y contraseña.

3. Envío de usuario y contraseña: una vez introducidos los datos de usuario y contraseña, el cliente envía la información al servidor. Para ello vuelve a repetir la petición inicial, pero enviando esta vez una cabecera extra con la información de autenticación. Esta información se generará codificando en *Base64* una cadena que incluye el nombre del usuario, dos puntos y la contraseña. Por ejemplo la cadena *user:password* convertida quedará de esta manera: dXNlcjpwYXNzd29yZA==

4. Respuesta del recurso: el servidor recibe la nueva petición de página con la información de autenticación. Si la información suministrada es válida, se permite el acceso al recurso, y, en caso contrario, se vuelve a responder con un código de error 401, para que el usuario vuelva a introducir su información. En el caso de que la autenticación haya sido positiva, en las siguientes peticiones de página se enviará automáticamente la cabecera con la autenticación, para evitar que el diálogo que pide el nombre de usuario y la contraseña aparezca continuamente.

En nuestro caso, puesto que la conexión no se realiza desde un navegador sino que se hace mediante código, el usuario y contraseña debe enviarse como una cabecera de la petición, con lo cual, una vez tenemos dos objetos de clase *String* que contienen la *URL* de conexión y la

autenticación respectivamente, el código para acceder a los datos sería el siguiente:

```
try {
	HttpClient httpclient = new DefaultHttpClient();
	HttpGet httpget = new HttpGet(url);
	String usuarioBase64 = getUsuario();
	httpget.setHeader("Authorization", "Basic " + usuarioBase64);
	HttpResponse response;

	try {
		response = httpclient.execute(httpget);
		HttpEntity entity = response.getEntity();

		if (entity != null) {
			InputStream instream = entity.getContent();
			//...
		}
	} catch (ClientProtocolException e) {

	} catch (IOException e) {

	}
} catch (Exception e) {

}
```

Como se ha comentado antes, en el apartado *5.5 Otras tecnologías utilizadas* se explicará cómo obtener el *usuarioBase64* y qué hacer con el *instream*.

Para poder realizar accesos a Internet ha de añadirse el correspondiente permiso al archivo de manifiesto de la aplicación:

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

5.3.2.5 Activar servicio de búsqueda en un Activity

En el apartado *Anexos* puede consultarse los botones de los que dispone un dispositivo *Android*. En él se explica el funcionamiento del botón de búsqueda, el cual si queremos que funcione en nuestra aplicación debe ser activado. En este apartado se explica cómo hacerlo.

Lo primero de todo es crear un archivo de configuración para permitir realizar la búsqueda, el cual tendrá que ser asociado posteriormente a todas las actividades que vayan a implementar una búsqueda. Esto hará que, cuando en una pantalla el usuario pulse el botón de búsqueda, la barra de búsqueda aparezca con el icono de la aplicación para que el usuario sepa que, en este caso, el texto a buscar se realizará sobre la pantalla que está viendo; en caso contrario se realizará la búsqueda por defecto que tenga asociado el dispositivo (contactos, Internet...). Un ejemplo de las diferencias entre una actividad con la búsqueda activada y otra desactivada lo observamos en las Figuras 14 y 15. Vemos que, como la ventana del listado de clientes sí tiene activada la búsqueda, nos deja en esa pantalla y muestra el icono, mientras que, en el segundo caso que es en la pantalla de la Agenda, al no estar activada simplemente nos lleva a la ventana de búsqueda del dispositivo.

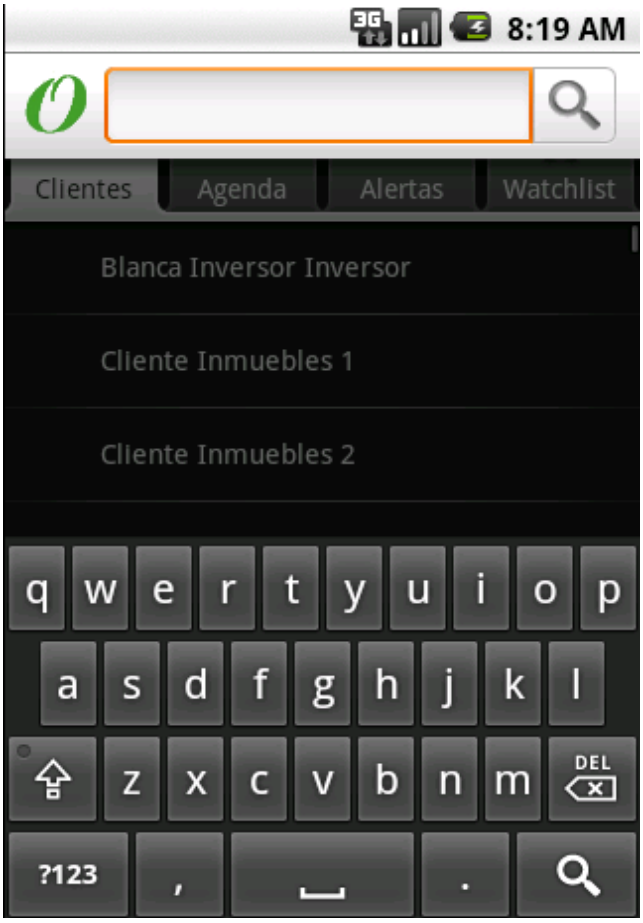


Figura 14: Actividad con búsqueda activada

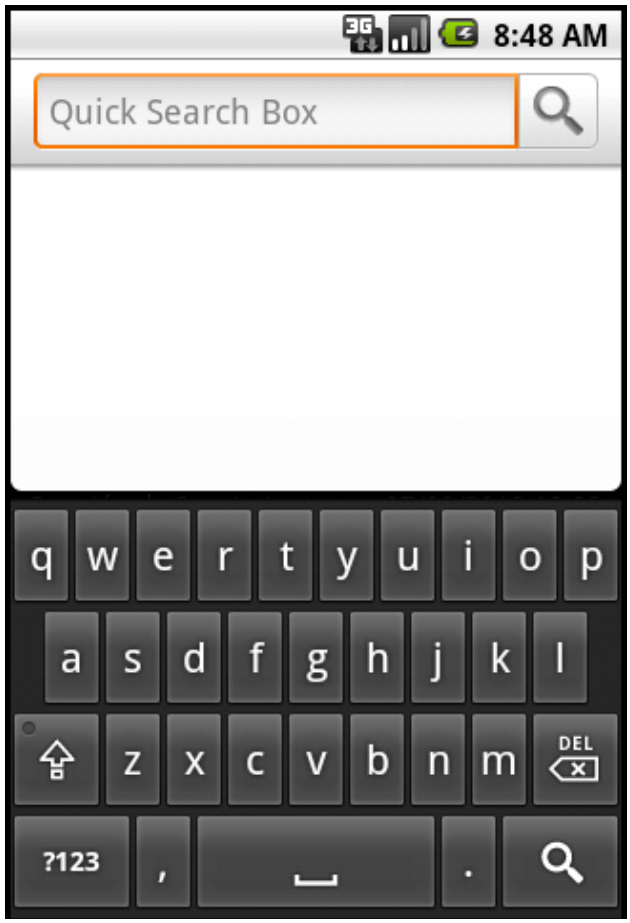


Figura 15: Actividad con búsqueda desactivada

El archivo de configuración de búsqueda debe ser un archivo *XML* que guardemos en alguno de los directorios de la carpeta *Res*. *Android* aconseja que se llame *searchable.xml* y se almacene en la carpeta *xml*. Su contenido debe ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
      android:label="@string/app_label" >
</searchable>
```

El siguiente paso es declarar las clases que queramos como *searchables* y asociarles el archivo que acabamos de configurar. Esto se hace en el archivo *AndroidManifest*. Tal y como se ha explicado anteriormente, donde debemos asociar las siguientes líneas marcadas en rojo:

```
<activity android:name=".ListaClientes"
            android:label="@string/app_name">
      <intent-filter>
            <action android:name="android.intent.action.SEARCH" />
      </intent-filter>
      <meta-data android:name="android.app.searchable"
             android:resource="@xml/searchable"/>
</activity>
```

En el caso del atributo *android: resource="@xml/searchable"* debemos poner la ruta dentro de la carpeta *Res* donde hemos guardado el fichero mencionado anteriormente. En este caso se llama *searchable.xml* y está en la carpeta *xml* dentro del directorio *Res*.

El último paso es saber cuándo se ha activado la búsqueda en la actividad y tratarla. Al igual que cuando es cambiada la orientación de la pantalla, cuando se finaliza la búsqueda la actividad es destruida y se vuelve a ejecutar el *onCreate()*. Por esta razón tenemos que introducir el siguiente código en ese método para recuperar el texto introducido para la búsqueda:

```
//Devuelve el Intent que lanzó la actividad
Intent intent = getIntent();
//ENTRA AQUÍ CUANDO SE HA SOLICITADO UNA BÚSQUEDA
if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
      //Texto introducio por el usuario
      String busqueda = intent.getStringExtra(SearchManager.QUERY);
      //Tratamiento de los datos de búsqueda
      //...
}
```

5.3.2.6 Realizar una llamada

Este caso se ha explicado anteriormente y es muy sencillo, simplemente hay que construir un *Intent* para realizar esa acción y pasarle en una variable de tipo *String* el número al que se desea llamar. Esto se hace de la siguiente manera:

```
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:" +
      numeroTel));
startActivity(intent);
```

En el archivo de manifiesto deberemos añadir el correspondiente permiso para realizar esta acción:

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

5.3.2.7 Enviar un SMS

Para realizar esta acción es necesaria la clase *SmsManager* del API y añadir el siguiente permiso en el archivo de manifiesto:

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

Según la API y la mayoría de los libros y tutoriales consultados, una vez tenemos el número destinatario y el texto a enviar, es suficiente con estas dos líneas de código:

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage(numeroDestinatario, null, texto,
      null, null);
```

Donde los campos puestos a *null* son respectivamente los siguientes: centro de servicio de mensajería (el cual siempre he encontrado *null* en todos los ejemplos), *Intent* que se encargará de enviarlo y por último el *Intent* que se encargará de recibir respuesta. En caso de que estos campos estén a *null* se usan los valores por defecto.

Como se ha comentado, es una API reciente y que es mejorada día a día, pero en la que aún quedan fallos. En este caso, y habiéndolo probado de varias maneras, este código siempre envía al menos dos SMS, incluso cuando el tamaño del texto es inferior al tamaño máximo de un SMS.

Afortunadamente la clase *SmsManager* cuenta con otro método para el envío de mensajes, y que en algunas de las fuentes consultadas aconsejan usar siempre ese. El método es *sendMultipartTextMessage()* cuyos parámetros son los mismos que en el caso anterior, sólo que los tres últimos parámetros deben ser un *ArrayList*. Es decir, en vez de un *String* con el texto se pasa un *ArrayList<String>* en el que el texto se haya dividido en partes (ahora se explicará cómo), y en vez de un *Intent* para envío y recepción pasar *ArrayList<Intent>*. Se aconseja usar este método aunque se controle que el texto a enviar no exceda el tamaño máximo de un SMS, ya que siempre funciona correctamente. El código quedaría de la siguiente manera una vez tenemos almacenado en variables el número del destinatario y el texto a enviar:

```
SmsManager smsManager = SmsManager.getDefault();
ArrayList<String> mensajes =
      smsManager.divideMessage(texto);
ArrayList<PendingIntent> listOfIntents = new ArrayList<PendingIntent>();
for (int i=0; i < mensajes.size(); i++){
      Intent sentIntent = new Intent("openfinance.workplace.Enviar");
      sentIntent.putExtra("SMS_ADDRESS_PARAM", destinatario);
      sentIntent.putExtra("SMS_DELIVERY_MSG_PARAM", (mensajes.size() > 1)?
            "Parte " +  i + " de SMS " : "SMS ");
      PendingIntent pi = PendingIntent.getBroadcast(Enviar.this, 0,
            sentIntent, PendingIntent.FLAG_CANCEL_CURRENT);
      listOfIntents.add(pi);
}
smsManager.sendMultipartTextMessage(destinatario, null, mensajes,
      listOfIntents, null);
```

5.3.2.8 Enviar un correo electrónico

Este caso es más sencillo que el anterior y una vez tengamos guardado en variables de tipo *String* la dirección de correo destinataria, el texto del mensaje y el texto del asunto del email, el código para enviarlo será el siguiente:

```
Intent emailIntent = new Intent(android.content.Intent.ACTION_SEND);
emailIntent.setType("plain/text");
emailIntent.putExtra(android.content.Intent.EXTRA_EMAIL, new String[]{
      destinatario});
emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT,
      textoAsunto);
emailIntent.putExtra(android.content.Intent.EXTRA_TEXT,
      textoEmail);
startActivity(Intent.createChooser(emailIntent, "Seleccione servidor de
correo"));
```

Vemos en este código que, a diferencia del SMS, aquí el parámetro para el destinatario debe pasarse como un vector de *String* aunque este solo sea uno. Para mandar el mismo SMS a varios destinatarios debería ponerse el código en un bucle e ir actualizando el destinatario en cada iteración.

También es destacable la forma de lanzar este *Intent*. Esto es debido a que el SMS se enviará sin más, pero en este caso el *Intent* lo que hace es mostrar un listado con las aplicaciones de correo electrónico que el usuario tenga instalado (por defecto tendrá como mínimo *Gmail*) y una vez

seleccione una, se abrirá esa aplicación con los campos completados con la información que le hemos pasado (destinatarios, asunto y texto) para que lo envíe desde allí.

Esta diferencia se debe a que, para un SMS, el número emisor es univoco ya que el teléfono tiene uno y siempre sólo uno; sin embargo, para un correo electrónico puede que el usuario no haya asociado ninguna de sus cuentas al dispositivo o a la aplicación de correo. Es decir, puede tener cero, una o varias. De esta forma seleccionará desde cual mandarla.

Puesto que el envío final lo hace otra aplicación no es necesario ningún permiso para realizar esta acción, simplemente deberá estar conectado a Internet cuando lo vaya a enviar desde la aplicación seleccionada.

5.3.2.9 Crear una pantalla de configuración

El programador tiene dos opciones si es necesaria una pantalla de configuración. Por un lado puede crearla a su gusto como una actividad más y almacenar los datos como desee. Por otro lado la API ofrece una forma de construir esta ventana de forma automática y mostrando una apariencia similar a la de todas las ventanas de configuración del dispositivo.

Otra de las ventajas de esta última opción es que el programador no debe preocuparse por el almacenamiento y recuperación de datos ya que se encarga de ello cada vez que se abre la ventana de configuración. No obstante se puede seguir teniendo acceso a estos datos y manejar esta pantalla como una actividad más aunque se realice de esta manera. A continuación se explica cómo construir esta pantalla.

Serán necesarios dos archivos, un archivo *XML* en la carpeta *Res* en el subdirectorio que queramos y con el nombre que deseemos; otro en la carpeta del directorio *SRC* donde tengamos el resto de las clases *Activity*, sólo que en este caso en vez de *Activity* la clase heredará de *PreferenceActivity*.

Por ejemplo en el directorio *Res/xml/* creamos el archivo *configuración.xml*, y para esta aplicación en concreto su contenido es el siguiente.

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:key="configuracion" android:title="Configuración"
      android:summary="Establecer opciones de configuración">

      <PreferenceCategory android:title="Configuración PIN"
            android:key="configurar_pin">
            <CheckBoxPreference
                  android:title="PIN activo"
                  android:summary="Establece si solicitar PIN al inicio de
                        la aplicación"
                  android:defaultValue="false"
                  android:key="@string/acceso_pin">
            </CheckBoxPreference>
            <Preference
                  android:key="@string/acceso_cambiar_pin"
                  android:summary="Modificar PIN actual o establecer uno
                        si todavía no existe"
                  android:title="Modificar PIN">
            </Preference>
```

```
        </PreferenceCategory>

        ...

</PreferenceScreen>
```

Este código puede ser construido con el editor gráfico que proporciona Eclipse, esto se mostrará más adelante en el apartado *5.3.3* C*onstrucción de interfaces gráficas*. Ahora se listan los elementos que pueden aparecer en este archivo, los cuales son muy parecidos a los *View* mencionados anteriormente.

- *PreferenceScreen*: similar al *Layout* pero, para la ventana de configuración, puede agrupar otros objetos.
- *PreferenceCategory*: permite dividir la ventana en submenús, es otro objeto contenedor.
- *CheckBoxPreference*: similar al *CheckBox,* solo que en este caso cuenta con un texto y un texto explicativo.
- *EditTextPrerence*: tiene un título y un texto explicativo, y cuando se pulsa aparece una ventana de diálogo con un campo para editar texto.
- *ListPrerence*: similar al *Spinner*, pero con título y texto descriptivo.
- *Preference*: el *Preference* más básico, se le puede asociar título y descripción y la acción que se desee. En este caso cuando se le hace clic nos lleva a otra pantalla.

En la Figura 16 se puede ver el resultado del fichero *XML* presentado anteriormente, con una explicación de que tipo de clase es cada objeto.

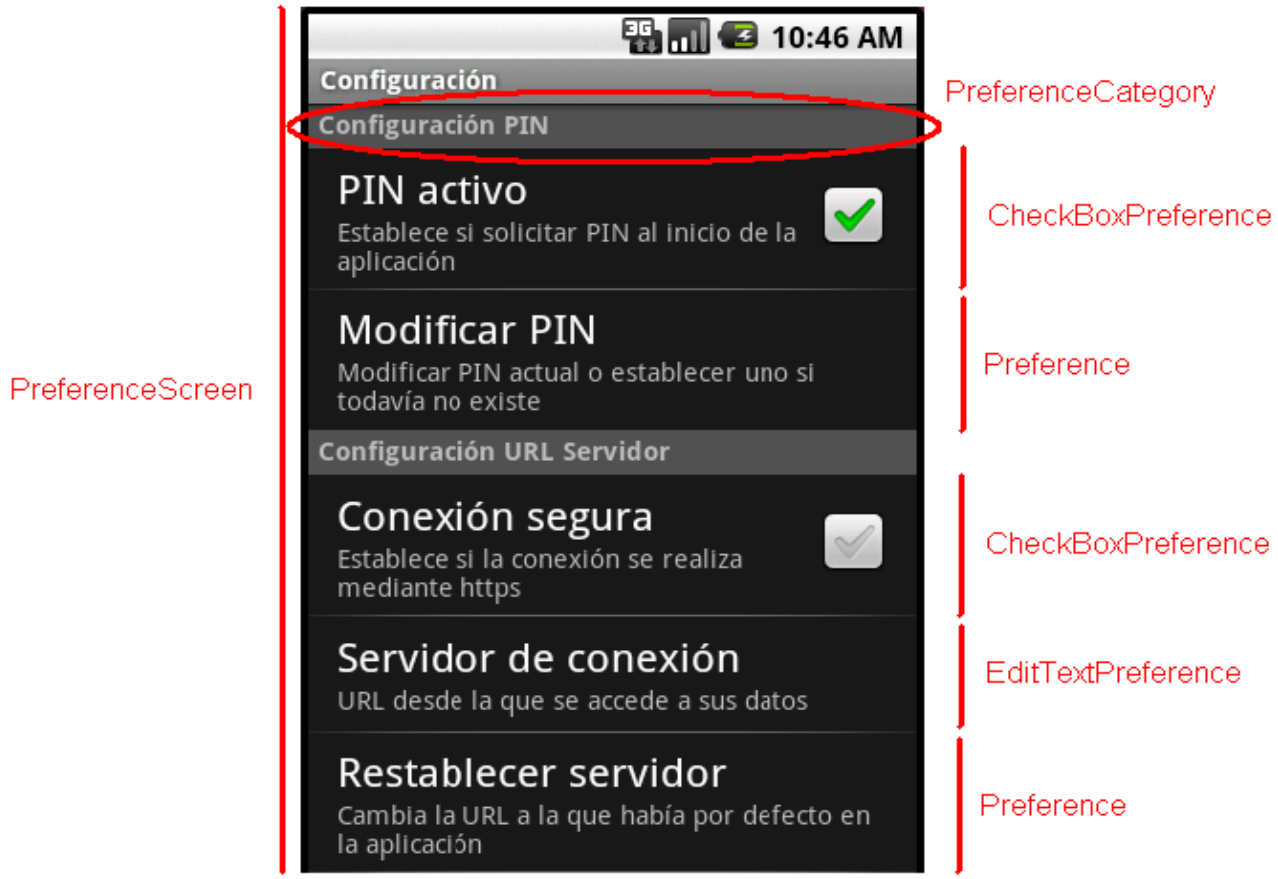


Figura 16: elementos de la ventana de configuración

Una vez hemos terminado el fichero XML tendremos que crear una clase que herede de *PreferenceActivity* para mostrarla por pantalla, esta clase puede implementar el resto de métodos y acciones que se han descrito hasta ahora y deberá declararse en el archivo de manifiesto.

Una diferencia con respecto a las actividades descritas anteriormente es la forma de acceder al *XML*. En un ejemplo anterior, se explicó como asociar un fichero que define una interfaz a una actividad. En este caso la forma de asociar actividad e interfaz es la siguiente:

```
public class Configuracion extends PreferenceActivity {

	private CheckBoxPreference checkBoxPin;
	private CheckBoxPreference checkBoxServidorSeguro;
	private EditTextPreference urlServidor;
	private Preference modificarPIN;
	private Preference restablecerURL;

	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		addPreferencesFromResource(R.xml.configuracion);
		checkBoxPin = (CheckBoxPreference)
			findPreference(getResources().getString(
			R.string.acceso_pin));
		checkBoxPin.setOnPreferenceChangeListener(
			manejadorCeckBoxPin());
		//...
	}
	//...
}
```

Con las dos primeras líneas del método *onCreate()* sería suficiente para que la pantalla de configuración se muestre y maneje los datos de forma automática. Pero como se ve en el código podemos también acceder a los objetos que definimos en el fichero *XML* para realizar tratamientos especiales con ellos si así lo deseamos. Esto se hace a través del método `findPreference()` al que tenemos que pasar como argumento el *String* que le diésemos en el valor *key*.

5.3.2.10 Almacenamiento persistente de datos

Se ha hablado al explicar la creación de la pantalla de configuración que ésta se encarga de almacenar y recuperar automáticamente los datos que en ella se muestran. Si el usuario desea acceder a estos datos o almacenar otros, estas son las opciones que la API de *Android* proporciona para el almacenamiento de datos:

- Estado de la aplicación: *Android* proporciona una serie de métodos que permiten guardar estados o el objeto que se desee ante determinados eventos, como por ejemplo el caso visto al cambiar la orientación de la pantalla.

- Ficheros: permite la escritura y lectura de ficheros, tanto en el directorio en el que se encuentra instalada la aplicación, como en la tarjeta de memoria.

- Bases de datos: cuando los datos a almacenar son abundantes o siguen una estructura más compleja se cuenta con la opción de construir una base de datos *SQL* a través de la librería *SQLite*.

- *SharedPreferences*: permite crear archivos en los que almacenan datos en la forma atributo/valor. Es la forma de almacenamiento usada para esta aplicación, y es explicada en profundidad a continuación.

Los archivos de preferencias son idóneos cuando se tienen que almacenar sólo unos pocos datos y sin estructuras muy complejas. Además, permite distintos niveles de seguridad, desde modo público en el que otras aplicaciones pueden acceder a estos datos (el cual no es público en si, ya que el programador de esa aplicación debe conocer el sistema de directorios creado por la otra aplicación así como el nombre del fichero de preferencias y cada atributo para acceder a los datos) a modo privado en el que sólo la aplicación que lo ha creado tiene acceso.

Como se ha comentado, los datos se almacenan de la forma atributo/valor. La forma de trabajar es parecida a la de un *HashMap,* pero realmente lo que está generando el objeto *SharedPreferences* es un archivo *XML*. Se muestran dos formas de acceder a un archivo de preferencias: la primera es para acceder al archivo de preferencias que la aplicación tiene asociado por defecto; es decir, el que ha creado automáticamente la ventana de configuración. El segundo es crear nosotros todos los que queramos. Este es el código para cada caso:

1. Archivo de configuración creado automáticamente

```
PreferenceManager.setDefaultValues(context, R.xml.configuracion, false);
SharedPreferences preferencias =
      PreferenceManager.getDefaultSharedPreferences(context);
```

context es un objeto de la clase *Context*, y esta debe ser la actividad que ejecuta el método. Cuando la clase hereda de *Activity*, *context* es *NombreClase.this* como se ha visto en la creación de los *Intent*.

2. Archivo creado por el programador

```
SharedPreferences preferencias = getSharedPreferences(String nombre,
      Context.MODE_PRIVATE);
```

En este ejemplo accede al fichero de preferencias cuyo nombre sea el que pasamos como primer atributo. En caso de no existir lo creará en ese momento con el grado de privacidad que se le pase en el segundo atributo.

Una vez se tiene el objeto de preferencias, accediéndose de una manera o de otra, la forma de obtener sus datos es la siguiente:

```
//SI NO EXISTE LA ENTRADA DEVUELVE EL VALOR DEL SEGUNDO CAMPO
String s = preferencias.getString(String nombreAtributo, null);
```

Este método devolverá el valor asociado al atributo del archivo de preferencias cuyo nombre es pasado como primer argumento. En caso de que este atributo no exista devolverá el valor pasado como segundo argumento.

Los objetos de preferencias no sólo trabajan con *Strings*. Se pueden almacenar y recuperar el resto de tipos básicos: *boolean*(*getBoolean()*), *int* (*getInt()*), *float* (*getFloat()*)...

Si lo que se quiere ese añadir o modificar el valor de un atributo se hace de la siguiente manera:

```
Editor editor = preferencias.edit();
editor.putString(String atributo, String valor);
editor.commit();
```

5.3.2.11 Comprobar estado de la conexión

En la mayoría de ventanas de esta aplicación se hace uso de conexión a Internet para listar acceder a los datos. Así siendo, un aspecto importante es conocer si existe la posibilidad de conexión antes de intentar acceder a ellos.

Por ejemplo se comprueba la conexión cuando se inicia la aplicación y si no se dispone de ningún tipo de acceso a Internet (Wi-Fi o red móvil), no se dejará iniciar la aplicación hasta que haya conexión. Esta comprobación se vuelve a realizar cada vez que una actividad necesite leer datos de Internet.

Para poder hacer la comprobación lo primero de todo es añadir el siguiente permiso en el archivo de manifiesto:

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

La API proporciona la clase *ConnectivityManager* que nos permite entre otras cosas comprobar el estado de los distintos tipos de conexiones. Un teléfono puede disponer de varios tipos de conexión según su modelo; en este caso se comprueba la conexión a cualquiera de los dos tipos de conexión de cualquier *Android.* En la aplicación construida este método se encarga de ello:

```
public static boolean comprobarConexion(Context context){
      ConnectivityManager connec = (ConnectivityManager)
            context.getSystemService(Context.CONNECTIVITY_SERVICE);
      NetworkInfo redWifi =
            connec.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
      NetworkInfo redMobile =
            connec.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
      if( redWifi.isAvailable() || redMobile.isAvailable() ) {
            return true;
      }
      return false;
}
```

En este caso se le pasa como parámetro un objeto de la clase *Context* porque el método no está definido en una clase *Activity*, con lo cual no puede ejecutar por si sola *getSystemService()*. Cada actividad que necesite comprobar la conexión llama a este método pasando como argumento *NombreClase.this* como se ha visto en otros casos.

5.3.2.12 Comprobar orientación de la pantalla

Anteriormente se ha explicado qué hacer para guardar los datos cuando el usuario cambia la orientación de la pantalla. Con este método podemos saber cuándo se ha producido esta acción, pero no en qué posición (vertical u horizontal) nos encontramos.

Conocer este hecho puede resultar muy útil para reutilizar el espacio en la posición horizontal. Por ejemplo, en esta aplicación, en las tablas que contienen muchos datos a mostrar y es importante su comparación, se muestra una columna más cuando la pantalla está en posición horizontal.

Dado que cada vez que este gire se vuelve a ejecutar el método onCreate(), es suficiente con comprobar la orientación en este momento o guardarla en una variable si se quiere utilizar posteriormente en más métodos. Esto lo podemos hacer con el siguiente método:

```
orientacion = getResources().getConfiguration().orientation;
```

En este caso el valor devuelto lo almacenamos en una variable de tipo int, para después utilizarla en otros métodos y mostrar más información o menos, según sea el valor de esta variable. La comprobación puede hacerse comparando con cualquiera de las dos siguientes variables que hacen referencia a posición vertical y horizontal, respectivamente:

```
//ESTA ES EL VALOR PARA POSICIÓN VERTICAL
android.content.res.Configuration.ORIENTATION_PORTRAIT


//ESTA ES EL VALOR PARA POSICIÓN HORIZONTAL
android.content.res.Configuration.ORIENTATION_LANDSCAPE
```

5.3.3 Creación de interfaces

Hay dos opciones para construir una interfaz en *Android,* definiéndola mediante código en la propia actividad o generándola en un fichero *XML* y accediendo a ella en el método *onCreate(),* como ya se ha explicado.

También se puede realizar una combinación de las dos, creando ciertos elementos que serán fijos en el fichero *XML* y añadiendo otros mediante código cuando sea necesario. Como ejemplo tenemos la construcción de tablas: definimos la tabla y sus atributos en el *XML,* y se van añadiendo filas conforme se lean los datos.

Se muestra un ejemplo de este caso para la ventana con la tabla de alertas. De esta forma se ven los dos casos: la creación de un elemento visual mediante *XML* y su acceso en el código, y también la creación de elementos visuales desde código y su forma de añadirlos para mostrarlos por pantalla. En el fichero *alertas.xml* se definen en su interior, entre otras cosas, dos *TableLayout* (una para carteras y otra para productos) con las siguientes características:

```
<TableLayout
      android:layout_width="fill_parent"
      android:id="@+id/TableLayoutCarteras"
      android:stretchColumns="1"
      android:layout_height="wrap_content">
</TableLayout>
```

Se muestra el código que define la tabla para mostrar las alertas de tipo carteras; respecto a la que contiene las de tipo productos su definición es igual salvo que con otro *id* y ubicada en un lugar distinto dentro de la jerarquía de posiciones de *alertas.xml*. Para modificarlas en el código lo que debe hacerse es acceder a ellas mediante su identificador de la siguiente manera en *onCreate()*:

```
tablaCarteras = (TableLayout) findViewById(R.id.TableLayoutCarteras);
tablaProductos = (TableLayout) findViewById(R.id.TableLayoutProductos);
```

Una vez tenemos el respectivo objeto de tipo *TableLayout* instanciado, para añadirles filas basta con un bucle en el que en cada iteración se lea una alerta y añadamos una fila con su información al final de la iteración. Se muestra un resumen del código que añade filas a la tabla:

```
TextView titulo;
TextView valor;
TableRow tr;
Bitmap imagenAlerta = BitmapFactory.decodeResource(this.getResources(),
      R.drawable.alert_icon);
ImageView iconoAlerta;
//...
for(int i = 0; i < alertas.size(); i++) {

      alerta = alertas.get(i);

      tr = new TableRow(this);
      tr.setId(i);

      titulo = new TextView(this);
      titulo.setPadding(3, 5, 3, 5);
      titulo.setText(alerta.toString());
      titulo.setTextColor(Color.WHITE);
      titulo.setWidth(170);
```

```
        //tratamiento similar para valor pero con sus correspondientes datos

        iconoAlerta  = new ImageView(this);
        if( ( ! alerta.getFechaActivacion().startsWith(Fecha.FECHA_NULA)) &&
              alerta.getFechaActivacion().compareTo(fecha.getFechaInicio())
              <= 0 ) {
            iconoAlerta.setImageBitmap(imagenAlerta);
            iconoAlerta.setPadding(2, 10, 2, 0);
        } else {
            iconoAlerta.setImageBitmap(null);
        }

        tr.addView(iconoAlerta, new TableRow.LayoutParams(0));
        tr.addView(titulo, new TableRow.LayoutParams(1));
        tr.addView(valor, new TableRow.LayoutParams());

        if( alerta.getClase().equals(TIPO_CARTERA) ) {
              tablaCarteras.addView(tr, new TableLayout.LayoutParams());
        } else if( alerta.getClase().equals(TIPO_PRODUCTO) ) {
              tablaProductos.addView(tr, new TableLayout.LayoutParams());
        }

}
```

Para una mayor claridad del código se han omitido otras sentencias que aparecen en el flujo de datos implementado, pero en esencia este es el modelo a seguir para interactuar entre datos estáticos y dinámicos de interfaz. En cada iteración se instancia de nuevo cada elemento que queramos añadir y se le da valor a los atributos que deseemos. En el caso de *iconoAlerta* es un objeto de tipo *ImageView* en el que en cada iteración se la instancia con la imagen guardada en la carpeta *Res/drawable* para avisar de que una alerta está activa o se le pone como imagen *null* para que no aparezca nada.

El inconveniente tanto de crear interfaces por código como de hacerlo en el fichero XML es que no podemos ver su apariencia hasta que no ejecutamos la aplicación. Para subsanar este problema *Eclipse* cuenta con un editor visual de interfaces con el que, a través de una interfaz, podemos ir añadiendo elementos y darle valor a sus atributos. Esta interfaz, a parte de mostrarnos como se verá la ventana en tiempo de ejecución va generando el fichero *XML*. Esto se explica a continuación en el apartado *5.4 Android en Eclipse IDE*.

Otras de las opciones que tenemos es la creación de nuestras propias vistas. Esto es, definirnos una clase que herede de la clase *View* e implementarla a nuestro gusto, para posteriormente añadirla por código como si fuese un *Button*, *TextView* o cualquier otro elemento.

Por ejemplo, en esta aplicación se ha creado una clase *View* para representar gráficamente los datos de la distribución de una cartera:

```
private class GraficoQuesito extends View {
        //...
        public GraficoQuesito(Context context) {
        //...
        protected void onDraw(Canvas canvas) {
        //...
}
```

Y en el *onCreate()* que deseemos basta con esta línea para representarlo:

```
layoutGrafico = (LinearLayout) findViewById(R.id.LinearLayoutGrafico);
layoutGrafico.addView(new GraficoQuesito(this));
```

A partir de la versión 1.6, la API incluye una forma de definir los métodos asociados a una vista desde el fichero *XML*. Esto aporta diversas ventajas como mayor claridad de código y no tener que declarar e instanciar algunas de las vistas en código si su único uso es responder a eventos.

Por ejemplo, si un *Button* sólo lo vamos a utilizar para ejecutar cierto código cuando este sea pulsado, y no se desea cambiarle ningún aspecto de apariencia en tiempo de ejecución, podemos asociarle un método a ejecutar con la siguiente línea en el fichero *XML*:

```
<Button
      android:onClick="nombreMetodo"
      ...>
</Button>
```

Ahora simplemente debe definirse en la clase *Activity* que se asocie con el archivo XML en el que aparece este *Button* un método con ese nombre, el cual debe tener la siguiente visibilidad y parámetros:

```
public void nombreMetodo(View v) {


}
```

Por defecto esto solo funcionará si el elemento para el que definimos las propiedades es un *Button*. Si queremos que funcione con cualquier otro elemento visual debe ponerse con valor *True* la propiedad en la ventana de propiedades que se observa en la parte inferior de la Figura 19, o también puede hacerse añadiendo la siguiente línea en la vista de texto. Por ejemplo, aquí se muestra el código necesario para que un *TextView* responda a esta propiedad:

```
<TextView
      android:onClick="onClickTextViewColumnaTabla"
      android:clickable="true">

</TextView>
```

5.4 Android en Eclipse IDE

Las ventajas que ofrece programar en *Eclipse* son varias, y se van a ir explicando a lo largo de este apartado. Este entorno de desarrollo nos permite la creación, edición y ejecución de proyectos *Android* de una manera bastante sencilla.

Además el *plugin* de *Android* para Eclipse permite otras opciones ya explicadas como el acceso al *AVD Manager* que, entre otras cosas, sirve para crear dispositivos virtuales como se mencionó anteriormente, así como otras opciones que se explican al final de este apartado.

5.4.1 Creación de un proyecto Android

La creación de este tipo de proyecto es muy similar a lo de un proyecto *Java* habitual. La única diferencia son algunos de los valores que deben introducirse en el último paso de la creación, y que son los que se explican.

Para crearlo, una vez está abierto *Eclipse* debe pulsarse *File* → *New* → *Project*. Si ya está instalado todo correctamente deberá aparecer un directorio llamado *Android* dentro de la jerarquía de opciones. Desplegamos esta carpeta y seleccionamos *Android Project*. Esto nos llevará a la siguiente pantalla, que se muestra en la Figura 17, con los datos necesarios ya completados.

Los datos que se deben introducir son los que en la imagen aparecen señalados con un ovalo rojo a su alrededor, y estas con las características de cada uno de ellos:

- *Project name*: aquí debemos introducir el nombre que le queramos dar a la carpeta que creará Eclipse en el *workspace* para identificar el proyecto. Este nombre puede contener espacios si se desea.

- *Build Target*: en este recuadro aparece un listado con todas las versiones del SDK que tenemos instaladas. Debemos seleccionar aquella para la que deseamos implementar la aplicación teniendo en cuenta que, a mayor número de versión, dispondremos de más posibilidades e incluso nuevas clases a nuestra disposición, pero en todas las versiones anteriores a la seleccionada no se podrá ejecutar.

  Si este listado apareciese vacío, es porque alguno de los pasos explicados en el apartado *5.2 Especificaciones técnicas* no se ha ejecutado correctamente, con lo cual debe revisarse en profundidad todo este apartado para solucionarlo.

- *Properties*: aquí deben introducirse datos relacionados con la aplicación y su identificación posterior dentro del dispositivo.

  1. *Application name* es el nombre que le aparecerá al usuario en su menú de aplicaciones una vez esta haya sido instalada; puede contener espacios en blanco y otros caracteres.
  2. *Package name* es la jerarquía de directorios que se explicó anteriormente. Debe estar formado por al menos dos directorios, y ninguno de ellos puede contener espacios en blanco. En este ejemplo se tendrán tres carpetas.

3. *Create Activity*: si seleccionamos esta opción deberemos establecer qué nombre queremos para la actividad. Dentro del *Package name* establecido creará una clase *Java* que hereda de *Activity,* asociada a *main.xml* (la cual contiene un "Hello World") de la carpeta *Res/layout*, que también genera automáticamente si activamos esta opción. También declarará esta clase en el archivo de manifiesto y le pondrá los atributos necesarios para que sea la que se muestra al iniciar la aplicación.

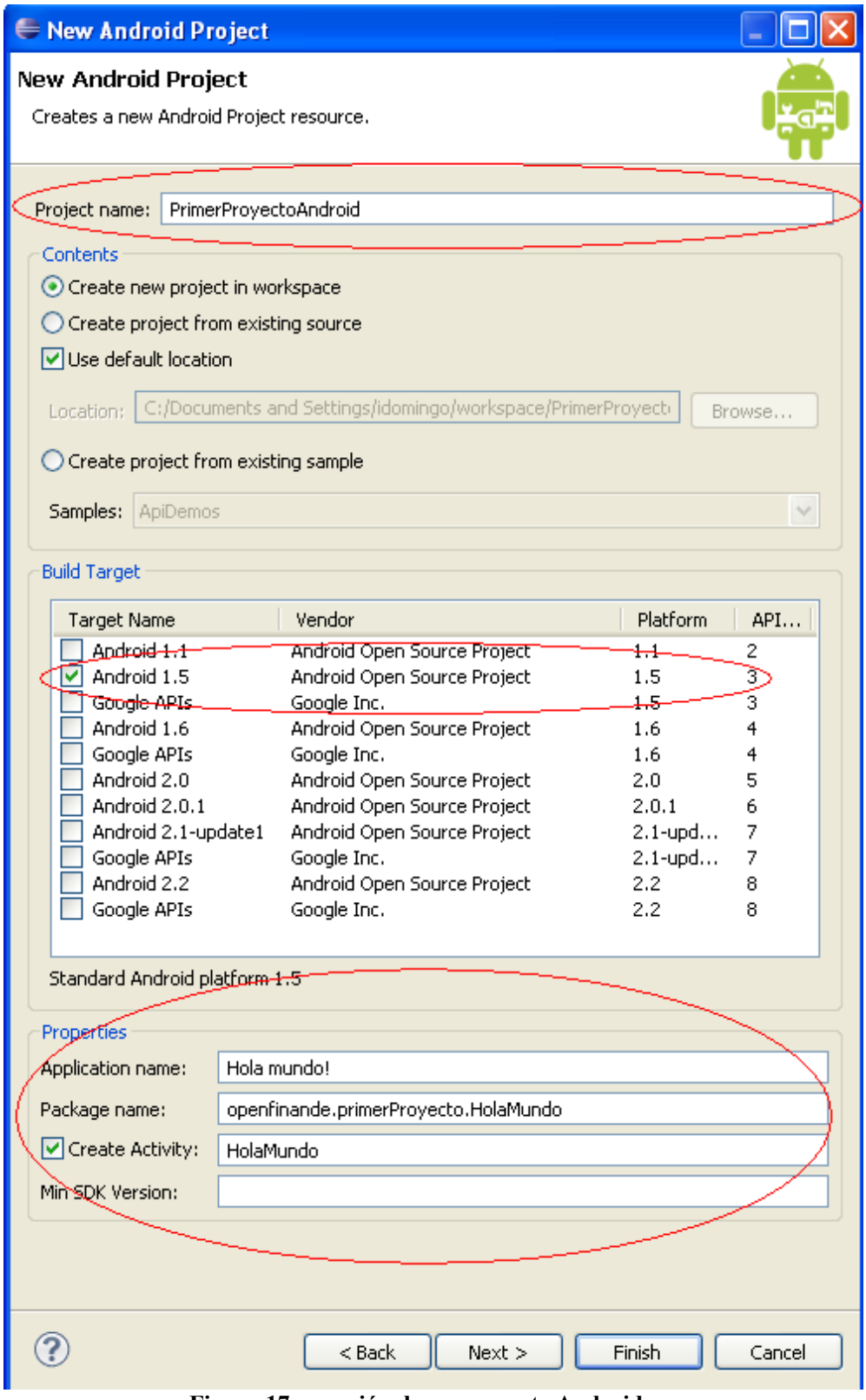


Figura 17: creación de un proyecto Android

5.4.2 Ejecución de un proyecto

La forma de ejecutar un proyecto *Android* es similar a la de cualquier proyecto *Java* en *Eclipse*. Para ello, sobre el nombre del proyecto se hace clic derecho con el ratón y dentro de las opciones seleccionamos *Run as* → *Android Application*.

Si hemos creado varios dispositivos virtuales y ninguno de ellos está abierto, seleccionará el primero de ellos que sea compatible para poder ejecutarse. Si tenemos ya abierto uno que sea compatible se ejecutará en éste; en caso contrario se abrirá uno en el que se pueda ejecutar. En caso de tener varios abiertos y que más de uno sea compatible con los requerimientos de esta aplicación, mostrará una ventana para que seleccionemos en cual ejecutar. Si, por el contrario, no existe ninguno en la aplicación que se pueda ejecutar, nos preguntará si deseamos crear uno.

5.4.3 Modificación de las características del proyecto

Es posible que consultando algún manual encontremos determinado método o clase del SDK de *Android* que nos convenga utilizar y, al usarlo en nuestro proyecto, no se permita. Tras comprobar que hemos escrito el código correctamente y hecho todo lo que diga el manual, si el error persiste es probable que se deba a que dicha clase o método pertenezca a una versión superior a la cual estamos trabajando.

Así siendo, si por este motivo o cualquier otro deseamos cambiar la versión de SDK que seleccionamos durante la creación del proyecto, esta puede ser cambiada en cualquier momento. Para ello se hace clic derecho sobre el proyecto y seleccionamos *Properties*. Esta acción nos mostrará la ventana que vemos en la Figura 18.

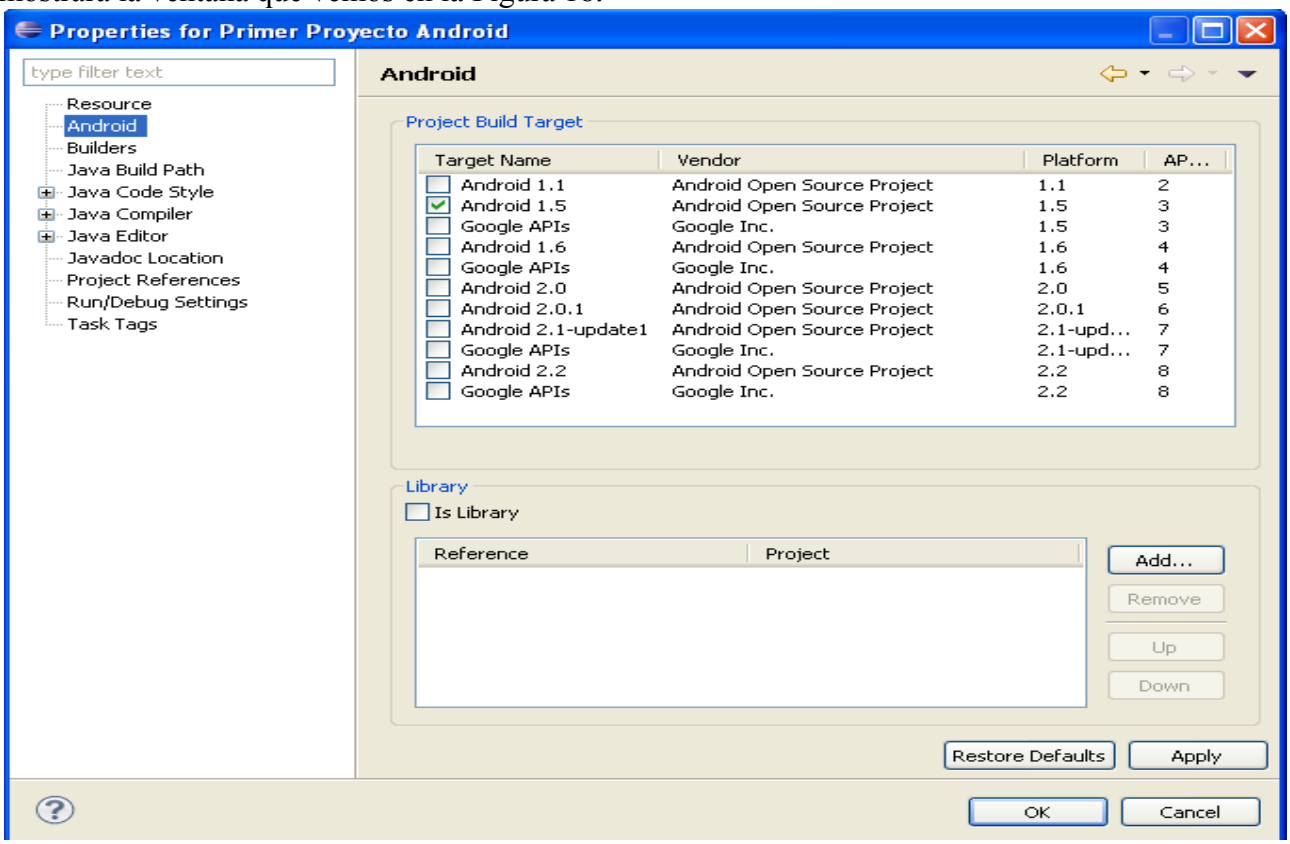


Figura 18: propiedades del proyecto

Si lo que se desea es cambiar el nombre de la aplicación, la versión mínima del SDK donde podrá ejecutarse, o el nombre del sistema de directorios, cualquiera de estas características se modifican directamente en el archivo *AndroidManifest.xml*

5.4.4 Creación de interfaces gráficas

Cada vez que abrimos un fichero de la carpeta *Res/layout* nos aparece una ventana como la de la Figura 19. En ella se puede alternar entre dos vistas como muestran las pestañas que observamos en el interior del óvalo rojo de menor tamaño, a la izquierda de la imagen. Si se pulsa en *alertas.xml* aparece la interfaz en formato texto, y si pulsamos en *Layout* aparece la ventana que se ve en la imagen. La explicación se realiza sobre esta, puesto que ya se ha hablado sobre la edición del *XML*.

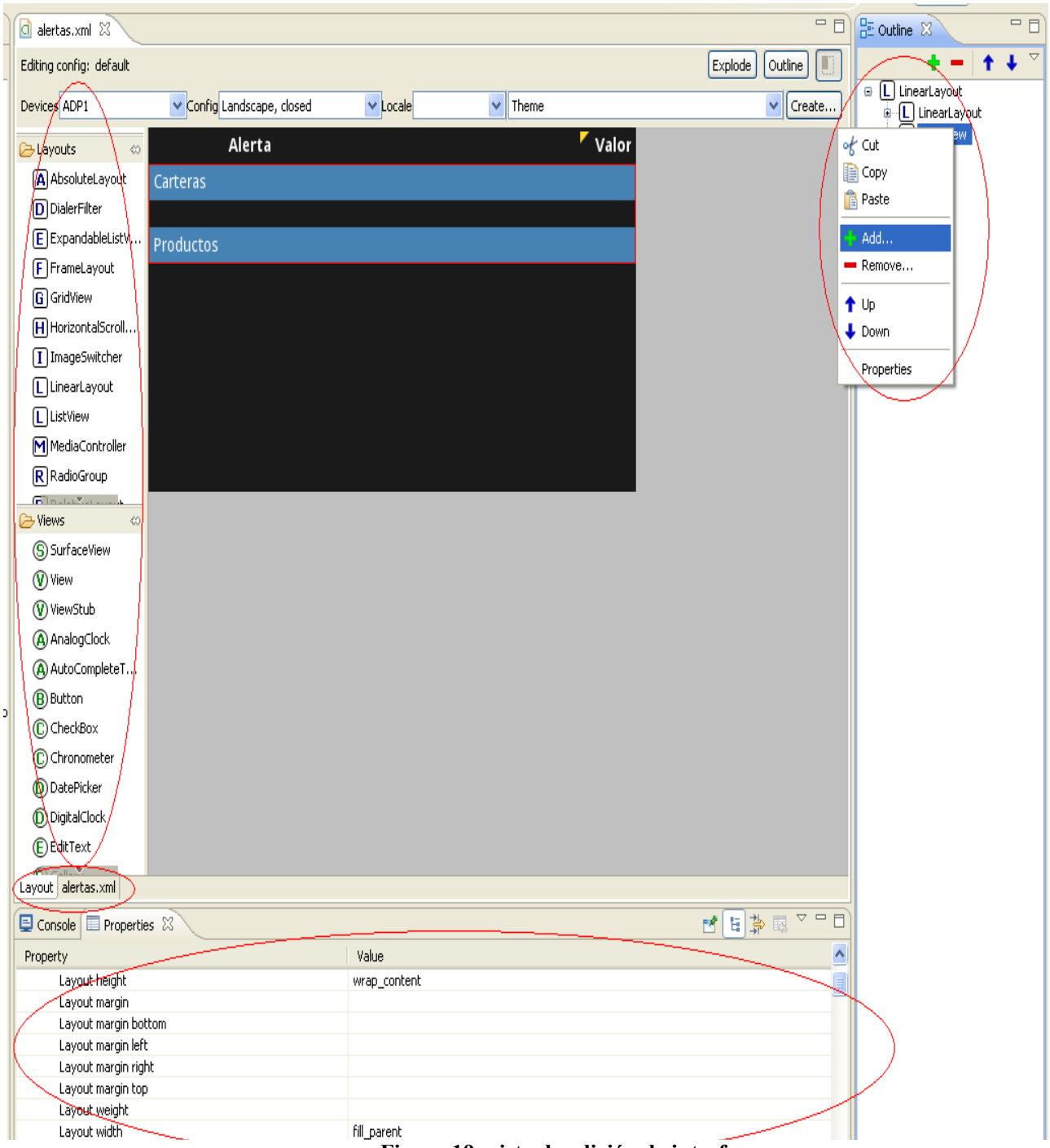


Figura 19: vista de edición de interfaz

En la imagen se han marcado los elementos más importantes de la vista *Layout.* En la parte izquierda aparecen todos los componentes que podemos añadir a la interfaz, divididos en *Layouts* y *Views*.

Para añadir cualquiera de ellos debe arrastrarse en la imagen sobre el *Layout* al que queramos asignarlo. En la parte derecha aparece nuestra jerarquía actual de componentes visuales para este archivo, la cual puede ocultar y desplegar el contenido de cada *Layout*. Si tenemos varios, en vez de arrastrar el objeto sobre el cual queremos añadirlo, es mucho más cómodo seleccionar en esta jerarquía aquel contenedor en el cual queremos añadirlo y, haciendo clic derecho sobre él, seleccionar *Add*. Esto nos mostrará un listado de todo lo que podamos añadir a ese contenedor, para que seleccionemos el componente deseado. En ambos casos el nuevo componente se posicionará en base a las características definidas del *Layout* en el que se introduce.

Cada vez que seleccionamos un componente del listado de la ventana derecha, en la parte inferior de la pantalla aparece su ventana de propiedades. En esta ventana podemos modificar sus atributos, lo cuales hacen referencia a opciones de visualización del propio elemento y a opciones de visualización de lo que contiene.

En caso de que cometamos algún error con este editor visual y queramos deshacerlo, sólo se puede hacer si pasamos a la vista de texto sin cerrar la ventana y así, se puede deshacer y rehacer en esta vista desde como estaba el fichero cuando se abrió.

Si marcase un error y en la vista de *Layout* no se viese nada, accediendo a la vista de texto marcará el error con una breve descripción en la línea que lo produce. Una vez no haya ningún error, todo volverá a mostrarse correctamente al acceder a la vista *Layout*.

Otra limitación de la vista de *Layout* es la reorganización de componentes. Aunque nos muestre un listado de la jerarquía actual, ésta no puede ser reordenada desde aquí. Las únicas modificaciones posibles son de las propiedades de cada elemento por separado. Puesto que el orden de aparición en pantalla es conforme a su aparición en esta jerarquía, si deseamos modificar el orden de visualización debe hacerse copiando y pegando en la vista de texto hasta que quede organizado como se desea.

5.4.5 Generar instalador

Este paso es muy similar a generar un archivo *.jar* con Eclipse cuando éste es un proyecto de *Java*. Pero, como se comentó anteriormente, Android usa *Java* pero no es una aplicación *Java*.

En este caso el instalador es un archivo con extensión .apk (*Android package*), el cual debe estar firmado digitalmente para poder ser instalado en un dispositivo. El certificado de firmas puede ser *self-signed;* esto quiere decir que no es necesario obtener un certificado digital de una entidad certificadora como pudiera ser por ejemplo *VeriSing*, sino que podemos hacerlo nosotros mismos mediante el siguiente comando de Java:

```
keytool -genkey -v -keystore "C:\android\certificado\keystore" -alias androidfirma -storepass password -keypass password -keyalg RSA -validity 14000
```

Donde éstos son los siguientes campos que el desarrollador debe modificar para poner sus propios datos puesto que éstos son de ejemplo:

- *keystore*: ruta donde se desea que se almacene el *keystore*.
- alias: nombre identificador con el cual luego nos referiremos a las claves que estamos creando.
- *storepass*: la clave para acceder al contenido del *keystore*.
- *keypass*: la clave para acceder posteriormente al alias que estamos creando.
- *keyalg*: tipo de algoritmo empleado, en este caso RSA.
- *validity*: periodo de tiempo de validez en días, se aconseja que sea de al menos 25 años. Para publicar la aplicación en el *Android Market* el certificado debe ser válido al menos hasta el 22 de octubre de 2033.

Este acción nos solicitará una serie de datos personales que asociará al alias que acaba de crear el *keystore*. Esto sirve para que, una vez firmemos el instalador, posteriormente pueda asociarse de forma univoca a la información dada aquí y no se pueda suplantar nuestra identidad. Para un *keystore* se pueden añadir tantos alias como se deseen volviendo a realizar este comando.

Una vez tenemos el fichero de firmas, el instalador puede ser firmado. Entre las diversas opciones está generar el *.apk* con *Eclipse* y firmarlo posteriormente usando, por ejemplo, *jarsigner* o generar un *.apk* firmado desde *Eclipse*. Ésta opción es la que se explica a continuación.

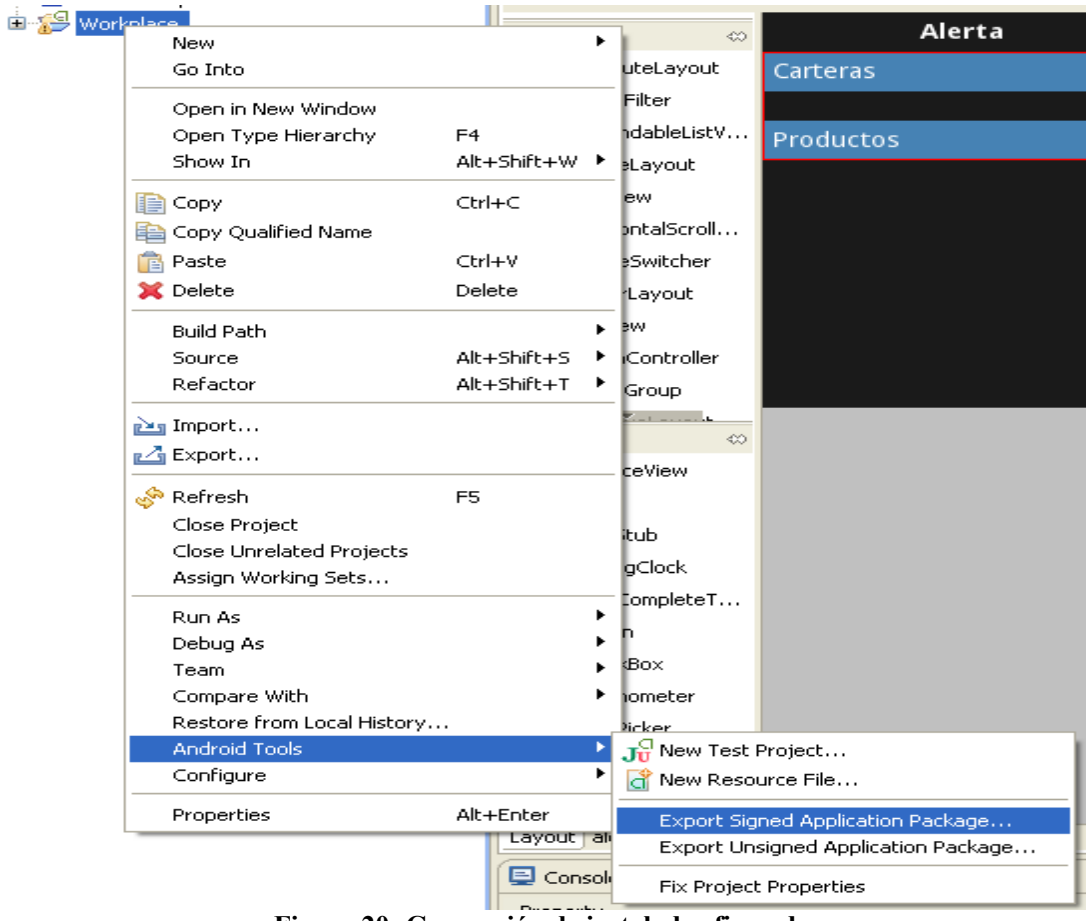


Figura 20: Generación de instalador firmado

Como muestra la Figura 20, haciendo clic derecho sobre el proyecto de entre las opciones disponibles seleccionamos *Android Tools*. Esto despliega varias opciones, de entre las cuales tenemos *Export Unsigned Application Package*, con la que sólo tendríamos que dar un nombre al archivo *.apk* y una ruta donde almacenarlo. Sin embargo es más interesante usar la opción *Export signed Application Package* ya que antes o después vamos a tener que firmarlo.

La primera ventana es para seleccionar el proyecto *Android* que deseamos firmar. Por defecto aparece el proyecto sobre el que hemos hecho clic con lo cual se puede pulsar en *next*. En la siguiente ventana debemos establecer la ruta en la que hemos generado anteriormente el fichero de claves y su contraseña de paso (valores dados en la línea de comando a -*keystore* y -*storepass*) y pulsar *next*.

Si la contraseña era correcta podremos seleccionar en la pestaña alias el identificador con el que queramos firmar. En caso de que solo haya uno aparecerá éste seleccionado, con lo cual sólo hay que poner la contraseña que asociamos a este alias (valor que asociamos a -*keypass*) y pulsar de nuevo *next*. Si esta última contraseña era incorrecta nos mostrará un error en la siguiente ventana. Si todo va bien deberemos establecer nombre y ruta para guardar el archivo *.apk,* el cual ya puede ser instalado en un dispositivo móvil.

5.4.6 Descargar proyectos de ejemplo

La página oficial de *Android* ofrece una gran cantidad de proyectos *Eclipse* de aplicaciones *Android*. Estos pueden ser consultados *online* para ver su código fuente o descargarlos para poder modificarlos y ejecutarlos.

La descarga puede realizarse automáticamente desde Eclipse de la siguiente manera:

- Abrir el *AVD Manager*: desde el botón en la barra de tareas de *Eclipse* o en el directorio en el que está instalado el SDK.

- En el panel seleccionamos *Available packages* y en el desplegable seleccionamos los que se deseen cuyo nombre empiece por "*Samples for SDK*" y pulsamos *Install Selected*.

- En la siguiente ventana se acepta la licencia y comenzará la descarga.

Esto creará en el directorio donde tenemos instalado el SDK un carpeta de nombre *Samples* donde estarán todos los proyectos que hemos descargado. Es una colección bastante completa en la que se incluyen aplicaciones tales como una agenda, un juego, chat a través de *Bluetooth*.... y una aplicación llamada *ApiDemos* existente para varias versiones del SDK que te muestra muchas opciones de lo que se puede realizar y cómo hacerlo. Es muy recomendable ejecutar esta aplicación y dedicarle tiempo si es la primera vez que se va a desarrollar para *Android*, puesto que te puede dar muchas ideas de interfaz y de que cosas puedes hacer.

Para abrir cualquiera de estos proyectos desde Eclipse debe hacerse lo siguiente:

File → New → Project → Android Project

Volviendo a la ventana de la Figura 17 explicada anteriormente deberíamos, en este caso concreto, seleccionar la opción *Create project from existing source* y establecer la ruta de cualquiera de los proyectos de ejemplo que hemos descargado.

5.5 Otras tecnologías utilizadas

Durante de la implementación del proyecto se ha hecho uso de distintas librerías externas tanto al JDK como al SDK que eran necesarias para llevar a cabo ciertas tareas, o en el caso de la lectura de ficheros *XML* para la obtención de un código más legible.

5.5.1 Autenticación *Basic*

Como se ha comentado en apartados anteriores, el sistema de autenticación empleado por este proyecto es *HTTP Basic*, con lo cual las credenciales enviadas al servidor tienen que estar codificadas en Base64.

Para ello se ha utilizado la librería *Apache Commons Codec,* que proporciona métodos para codificar y decodificar a *Base64*. Se trata de una librería gratuita que puede descargarse de la página oficial de Apache.

Una vez se ha descargado la librería, simplemente debe añadirse el fichero *.jar* al proyecto como librería externa y se realiza la codificación de la siguiente manera:

```
String usuarioBase64 = new
      String(Base64.encodeBase64(autentificacion.getBytes()));
httpget.setHeader("Authorization", "Basic " + usuarioBase64);
```

Donde *autentificacion* es un *String* que contiene el nombre de usuario y contraseña introducidos por el usuario ordenados de la forma usuario:contraseña.

5.5.2 Lectura de ficheros XML

Para el tratamiento de los datos que devuelven los servicios web se ha utilizado un tiempo una librería de *DOM4J* para la lectura de ficheros XML, que permite realizar un código más legible y mantenible que lo que se puede hacer con las clases proporcionadas por el SDK de *Android*. Esto se debe a que la API decidió no hacer uso de *XPATH* en las clases que proporciona para tratar este tipo de datos. De esta manera con DOM4J podemos procesar un fichero en un sólo método con unas pocas lineas de código, mientras que la API proporciona cinco métodos distintos que se explicaran a continuación.

Al igual que con la librería de Apache, se puede descargar de la página oficial de *DOM4J* y se añade al proyecto como librería externa, quedando la clase lectora de la siguiente manera:

```
abstract class LectorDOM4J {

	public ArrayList<Object> leer(InputStream datos) {

		ArrayList<Object> objects = new ArrayList<Object>();

		SAXReader reader = new SAXReader();

		try {
			Document document = reader.read(datos);
			procesarElementos(document.getRootElement(), objects);

		} catch (DocumentException e) {
			e.printStackTrace();
		}
		return objects;
	}

	protected abstract void procesarElementos(Element raiz,
		ArrayList<Object> lista);

}
```

Cada clase de la que necesitamos leer datos por Internet hereda de esta clase e implementa su método para procesar los datos conforme a los atributos que hay definidos en su archivo *XML*.

Esta librería permite definir todo el tratamiento en un método sencillo de pocas líneas, mientras que el problema de las clases proporcionadas por la *API* es que requieren muchas líneas de código ya que se ejecutan cinco métodos distintos para leer un fichero de datos siguiendo este esquema:

```
public void startDocument() throws SAXException {
	// Tratamiento al inicio del documento
}

public void endDocument() throws SAXException {
	// Tratamiento al final del documento
}

/** Se ejecuta cuando encuentra tags como:
* <tag>
* Puede leer atributos cuando se encuentra con:
* <tag attribute="attributeValue">*/
public void startElement(String namespaceURI, String localName,
		String qName, Attributes atts) throws SAXException {

}

/** Se ejecuta en tags de cierre:
* </tag> */
public void endElement(String namespaceURI, String localName, String
	qName) throws SAXException {

}
/** Se ejecuta cuando se encuentra con la siguiente estructura:
* <tag>characters</tag> */
public void characters(char ch[], int start, int length) {

}
```

Los dos primeros métodos sólo se ejecutarán una vez por fichero leído, pero los tres últimos se ejecutarán para cada atributo del documento y como se puede observar en los parámetros que recibe, hay que controlar en cada uno de ellos cual es el *tag* en el que nos encontramos, puesto que que el método que nos devuelve el contenido de un *tag* ni siquiera tiene parámetro para especificar en cual nos encontramos.

5.5.3 Acceso a *Google Maps*

Google ofrece una API propia que ofrece clases para usar varios de sus productos, como por ejemplo en este caso *Google Maps*. Esta API se descarga de la misma manera que la otra (con el *AVD Manager*) como se ha descrito en apartados anteriores.

En la Figura 17 puede observarse que existe la API de *Android Open Source Project* y la de *Google Inc*. que añade a ésta sus propias clases. Para poder utilizar las clases de acceso a los mapas, una vez se ha descargado la API de *Google* debe seleccionarse ésta para el proyecto, como se comentaba en el apartado 5.*4.3 Modificaciones de las características de un proyecto*.

Debe tenerse en cuenta que, ahora, para ejecutar este proyecto en el emulador deberá construirse un dispositivo virtual apropiado con el *AVD Manager*, como ya se ha explicado. Es decir, que contenga la versión del sistema operativo para la que estamos implementando.

En los otros casos explicados en este apartado, bastaba con añadir la librería como un *jar* externo del *project*; esto se podía ya que son librerías Java, no desarrolladas exclusivamente para trabajar en *Android*. Pero, en este caso, su uso debe determinarse de otra manera, añadiendo la siguiente línea en el archivo de manifiesto dentro del elemento *applicaction*:

```
<application
      android:icon="@drawable/icono_of"
      android:label="@string/app_name">
            <uses-library android:name="com.google.android.maps" />
      …
</application>
```

El único inconveniente de usar esta librería, es que si el usuario no tiene instalada la aplicación *Maps* en su dispositivo no podrá instalar la aplicación. Pero, al ser una aplicación de *Google Inc*. ésta es incluida por defecto en la mayoría de los terminales *Android*; además, esta línea que hemos incluido en el archivo de manifiesto avisará al usuario de que necesita esta aplicación si no la tiene. Por otro lado se pueden tener dos versiones del proyecto, puesto que con quitar esta línea del archivo de manifiesto y cambiar la API utilizada a cualquiera de las de *Android Open Source Project* sería suficiente para poder instalar la aplicación en cualquier dispositivo.

Si queremos de nuevo probar un proyecto de ejemplo tenemos uno, esta vez instalado ya cuando descargamos las APIs de *Google*. Para acceder a él, simplemente debemos acceder al directorio donde tengamos instalado el SDK. En la carpeta *add-ons* entramos en cualquiera de los directorios de *google_apis* y encontraremos una carpeta llamada *samples,* en cuyo interior está el proyecto de ejemplo *MapsDemos*.

Para poder ejecutar este proyecto de ejemplo o cualquier otro creado por nosotros que vaya a usar esta librería necesitamos obtener un *Maps API Key* que debemos introducir en el archivo *XML* que defina la actividad que usará *GoogleMaps*. Si no hacemos esto, la aplicación puede ser compilada y ejecutada, pero cuando vayamos a descargar la información de *GoogleMaps* para mostrarla por pantalla ésta no nos proporcionará nada.

Son necesarios dos elementos para poder obtener esta clave: un *keystore* (explicado cómo obtenerlo en el apartado 5.*4.5 Generar Instalador*) y una cuenta de *Gmail*. El *keystore* que utilicemos para obtener esta licencia debe ser el mismo con el que firmemos el instalador, puesto que en el momento de ejecución la aplicación *GoogleMaps* comprobará que sean el mismo o tampoco nos proporcionará información.

Cuando tengamos preparado el *keystore* con el que firmaremos la aplicación debe ejecutarse el siguiente comando para obtener una *huella digital de certificado MD5* (los datos puestos ahora son siguiendo el ejemplo dado anteriormente):

```
 keytool -list -alias androidfirma -keystore
"C:\android\certificado\keystore" -storepass password -keypass password
```

Una vez hemos ejecutado correctamente esta acción debemos guardar en un lugar seguro el certificado obtenido, o no cerrar la ventana de comando hasta que hayamos realizado todos los pasos. Ahora se debe entrar al siguiente enlace:

http://code.google.com/intl/es/android/maps-api-signup.html

Lo primero de todo en esta página es identificarnos con nuestra cuenta de *Gmail*. Después aceptamos la licencia e introducimos el certificado MD5 que acabamos de obtener, tras lo cual pulsamos en el botón *Generate API key.*

Si todo es correcto la siguiente ventana mostrará la clave, recordándonos que sólo será válida para aplicaciones firmadas con el *keystore* del que hemos obtenido el *certificado MD5*, y un código de ejemplo de lo que debemos hacer a continuación.

El siguiente paso a realizar es crear un archivo de definición de interfaz en la carpeta *Res/layout* para la actividad que acceda a *GoogleMaps,* y colocar la clave que nos acaban de proporcionar. Dicho XML puede ser este:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/mainlayout"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" >

      <com.google.android.maps.MapView
            android:id="@+id/mapview"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:clickable="true"
            android:apiKey="INTRODUCIR AQUÍ LA CLAVE"
      />

</RelativeLayout>
```

Se recuerda que el *keystore* con el que se obtiene el *apiKey* debe ser el mismo que con el que firmaremos la aplicación, con lo cual también ser recomienda obtener dos *apiKey:* una con el *keystore* que hemos creado, y otra con el que utiliza *Android* para ejecutar las aplicaciones en el emulador, puesto que si ponemos el *apiKey* obtenido con nuestro *keystore* en el emulador no funcionará el acceso a la información y viceversa. El *keystore* del emulador se encuentra en este directorio:

- Windows Vista: `C:\Users\<user>\.android\debug.keystore`
- Windows XP: `C:\Documents and Settings\<user>\.android\debug.keystore`
- OS X and Linux: `~/.android/debug.keystore`

La principal clase que *Google* nos proporciona es *MapView*, con lo cual ahora debemos crear una clave que herede de ésta y asociarla al archivo que acabamos de crear en la carpeta *Layout*.

Otra clase muy importante que nos proporciona esta API es *Geocoder*. *GoogleMaps* trabaja con longitudes y latitudes para su sistema de localización y esta clase nos permite dos formas de trabajar con estos datos:

- *Reverse geocoding*: a través de una longitud y latitud nos proporciona el nombre de esa calle.

- *Forward geocoding*: dada una dirección (cuanto más completa mejor) nos proporciona su latitud y longitud. Es la que utilizaremos en este caso y la que se explica a continuación, dado que lo que conocemos de cada cliente es su dirección.

Geocoder es una clase bastante potente y con tan solo un método en el que le pasamos un *String* con la dirección, y nos devuelve su ubicación. Evidentemente, cuanto más completa sea esta dirección, mejor lo localizará. No obstante, como segundo argumento debemos darle el máximo número de localizaciones posibles que queremos, puesto que si, por ejemplo, el *String* pasado es "avenida Aragón", probablemente encontrará tantas como le pidamos, mientras que si le pedimos que nos devuelva un máximo de cinco localizaciones, y el *String* dado es "avenida Aragón 46021, Valencia" sólo devolverá uno. Aquí se muestra un ejemplo del código utilizado:

```
Geocoder geocoder = new Geocoder(this, Locale.getDefault());
List<Address> localizaciones = null;
try {
    //A partir de una dirección, por ejemplo "Pintor Soroya 25, Valencia"
    //devuelve una lista de posibles localizaciones
    //el segundo argumento es el máximo de localizaciones que queremos
    localizaciones= geocoder.getFromLocationName(direccion, 5);
} catch(IOException e) {

}
```

Una vez tenemos una lista de objetos *Address* lo que debe hacerse es conseguir la longitud y latitud de cada una y pasársela a *GoogleMaps*, debe tenerse en cuenta que éste trabaja con estos valores en *micropulgadas*. El código utilizado para este proyecto es muy similar al ofrecido en el proyecto de ejemplo comentado anteriormente, pero, en este caso, se han añadido, entre otras cosas, más opciones, como por ejemplo presentar el mapa centrado en las direcciones obtenidas y con un nivel de *zoom* medio, como muestra el siguiente código:

```
for( Address a : localizaciones ) {
      //Latitud y longitud deben estar en microdegrees
      lat = a.getLatitude()*1E6;
      lng = a.getLongitude()*1E6;
      point = new GeoPoint(lat.intValue() , lng.intValue());
      mapController.setCenter(point);
```

```
            //1: Zoom Más lejano, 21: Más cercano
            mapController.setZoom(16);
            overlayitem = new OverlayItem(point, "", "");
            itemizedOverlay.addOverlay(overlayitem);
    }
```

5.5.4 Dibujar gráficas de evolución

Anteriormente se comentó que el desarrollador puede crear sus propias vistas. Como ejemplo de esto, se ha explicado que se ha creado la clase *GraficoQuesito,* que hereda de *View* y permite dibujar un gráfico de sectores. En este caso, para representar los datos de la evolución de una cartera, se ha optado por utilizar una clase desarrollada por terceros.

Ésta clase se llama *GraphView*, pertenece a un proyecto de código abierto que puede encontrarse en la siguiente URL, la cual ofrece una serie de tutoriales y proyectos de código abierto sobre *Android*:

http://android.arnodenhond.com/

Ésta vista nos permitirá representar gráficamente un vector de números, tanto de manera lineal como en forma de diagrama de barras. Para ello, al igual que en el ejemplo mostrado con la vista *GraficoQuesito*, llamamos al constructor de la clase y lo añadimos a la vista del *Layout* o actividad en la que se desea mostrar. Al cual hay que pasar los siguientes argumentos:

```
GraphView(Context context, float[] values, String title, String[] horlabels, String[] verlabels, boolean type)
```

Los cuales son: la actividad en la que se mostrará la vista, los valores a representar, el título de la gráfica, las etiquetas del eje X, las etiquetas del eje Y y el tipo de representación (*GraphView.LINE* o *GraphView.BAR).*

En este caso, se ha realizado alguna modificación sobre el código de esta clase. Por ejemplo, para adaptarse a los requerimientos de la gráfica, cuando el valor a representar es mayor o igual a cien, la linea se pintará de color verde y de color rojo en caso contrario, así como una línea horizontal amarilla que marca el valor cien.

Los resultados de ésta acción se podrán ver más adelante en el siguiente capítulo, concretamente en la Figura 50.

5.5.5 Generar un webservice

Los datos para representar la gráfica que se acaba de explicar son proporcionados por un servicio web *REST* que he desarrollado como ampliación de la aplicación, una vez esta estaba finalizada y validada.

El proceso y tecnologías empleadas para su elaboración ha sido el mismo que la empresa había utilizado para la implementación del resto de servicios web con los que conecta la aplicación. El lenguaje de programación que se ha utilizado es *C#* y la herramienta de desarrollo utilizada ha sido el *Visual Studio 2008*.

La idea de *REST* (*Representational State Transfer*) es aprovechar las capacidades del protocolo *HTTP* para utilizarlo como arquitectura de servicios. *REST* describe una interfaz web simple utilizando peticiones *HTTP* y datos *XML* pero sin capas adicionales como *SOAP*. Actualmente es la alternativa más simple a *SOAP* y a los servicios web basados en *WSDL* (*Web Services Descripcion Language*).

Al nuevo servicio web hay que asociarle una nueva *URI* para acceder a los recursos. Desde el punto de vista del cliente de la aplicación que accede a un recurso, la *URI* determina lo intuitivo que va a ser el servicio web *REST*, y si el servicio va a ser utilizado tal y como fue pensado en el momento que se diseñó.

Las *URI's* de los servicios web *REST* deben ser intuitivas, hasta el punto que sea fácil saber el recurso que nos van a proporcionar. Se piensa en las *URI* como una interfaz autodocumentada que necesita de muy poca o ninguna explicación o referencia para que un desarrollador pueda comprender a lo que hace referencia.

En este caso, como la información a obtener es sobre una cartera, la URI se ha implantado dentro de la jerarquía de los servicios web de las carteras y la URI resultante queda de esta manera:

```
http://OpenfinanceUrlBase/Carteras.svc/{idCliente}/carteras/
{idsCarteras}/evolucion?fechaIni={fechaIni}&fechaFin={fechaFin}
```

En este caso, idsCarteras puede ser el identificador de una cartera o el de varias carteras separadas por comas. En caso de varios identificadores, el servicio web crea una cartera virtual (llamada cartera agregada) formada por todas las carteras cuyo identificador se ha pasado como parámetro.

Otra acción a realizar es comprobar que el identificador del cliente es válido y que tiene permiso para acceder a cada uno de los identificadores de carteras. Finalmente, se devuelve la información con los datos de evolución de la cartera para el intervalo dado por *fechaIni* y *fechaFin*.

Si todos los datos son correctos el servicio web devolverá un XML con un valor para cada día, el primer valor siempre será cien, y los sucesivos valores siempre se calcularán a partir del valor del primer día. El fichero sigue la siguiente estructura:

```
<ListaEvolucion>
    <DatosEvolucion>
        <EvolucionData>
            <Dia>27/09/2009</Dia>
            <Valor>100</Valor>
        </EvolucionData>
        <EvolucionData>
            <Dia>28/09/2009</Dia>
            <Valor>100</Valor>
        </EvolucionData>
        <EvolucionData>
            <Dia>29/09/2009</Dia>
            <Valor>100.07798765033647</Valor>
        </EvolucionData>
        …
    </DatosEvolucion>
</ListaEvolucion>
```

6 Aplicación

6.1 Inicio de la aplicación

Una vez la aplicación ha sido instalada el usuario podrá acceder a ella desde el menú de aplicaciones de su dispositivo. La primera ventana que verá siempre es la ventana de inicio de la aplicación, la cual muestra el nombre y página web de la empresa. Esta ventana, además de ser informativa, sirve como compaso de espera puesto que, durante su visualización, se leen los datos de configuración y se comprueba la disponibilidad de conexión a Internet. Dicha pantalla se muestra en la Figura 21.

Figura 21: Pantalla de inicio

La siguiente ventana a mostrar puede ser una de las cuatro siguientes opciones, dependiendo de los datos leídos durante el periodo de carga:

- No se dispone de conexión a Internet: en ese caso se mostrará un mensaje informativo y se cerrará la aplicación, puesto que todas las pantallas necesitan acceso a Internet. La aplicación no se podrá iniciar hasta que la conexión esté activada. Esta ventana se mostrará más adelante en el apartado de mensajes de aviso.

– Es necesario PIN para entrar a la aplicación: si al leer los datos de configuración, la solicitud de PIN está activada se mostrará una ventana para que el usuario pueda introducirlo. En caso de que se solicite pero no haya ninguno almacenado, la ventana a mostrar será la de establecimiento de PIN como muestra la Figura 22. En caso contrario, se mostrará la ventana de la Figura 23, en la que debe introducirse el PIN existente; el usuario dispone de tres intentos, tras los cuales todos los datos de configuración serán borrados, y la próxima vez que inicie la aplicación el usuario deberá establecer un nuevo PIN, así como su nombre de usuario y contraseña en el *Workplace*.

– En esta pantalla los únicos datos que se pueden introducir son de tipo numérico. Teniendo en cuenta que se puede introducir un PIN incorrecto, y que se dispone de un máximo de tres intentos para hacerlo correctamente, se muestra un aviso en letras rojas, además de realizar una animación que "agita" la pantalla para que el usuario se dé cuenta inmediatamente de que los datos son erróneos.

Figura 22: Establecimiento de nuevo PIN

Figura 23: Ventana de introducción de PIN

– No hay nombre de usuario y contraseña almacenados: esto ocurrirá si es la primera vez que se ejecuta la aplicación, se introdujo el PIN incorrectamente tres veces, o se finalizó al hacer *logout*. Esta ventana muestra un formulario para introducir nombre y contraseña, y se accede a ella tras la ventana de presentación (o tras la de introducción de PIN si este estuviera activo) o cuando se hace *logout* en la aplicación. Al igual que en el caso descrito anteriormente, se produce una animación cuando la información introducida es errónea. Esta pantalla puede observarse en la Figura 24.

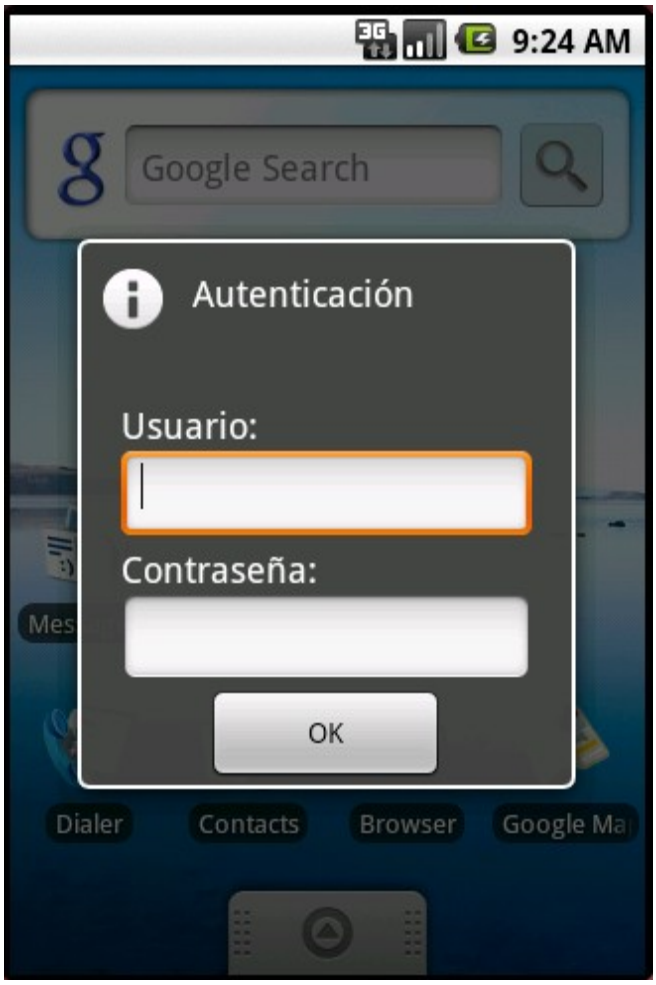


Figura 24: Ventana de autenticación

- No es necesario PIN de inicio, y hay nombre y contraseña almacenados. Este estado sólo puede ocurrir si en configuración se ha deshabilitado la solicitud de PIN para iniciar la aplicación y la última vez se cerró sin hacer *logout*. En este caso, tras la ventana de inicio, pasará directamente al menú de usuario, el cual se explica a continuación con más detalle.

6.2 Menú de usuario

Este menú está dividido en cuatro pantallas que muestran información referente al gestor, y a las cuales se puede acceder con las pestañas que aparecen en la parte superior de la ventana. A este nivel de navegación existen dos entradas de menú comunes a las cuatro pantallas – *logout* y configuración – y al resto de menús de la aplicación, los cuales serán detallados más adelante, pasando a continuación a explicar cada una de las ventanas con información del usuario.

La primera pantalla de este menú es el listado de clientes del gestor. En dicho listado se muestra por orden alfabético el nombre y apellidos de cada cliente, así como un icono con un teléfono en caso de que se tenga almacenado uno o más números de contacto del cliente. Todo esto se puede observar en la Figura 25.

Figura 25: Ventana del listado de clientes

Esta ventana permite realizar llamadas telefónicas a cada cliente. Para ello el usuario debe pulsar el icono del teléfono, el cual indica que existe algún teléfono de contacto almacenado en el *Workplace* para ese cliente. En caso de que sólo se disponga de un único número de contacto, se realizará automáticamente una llamada a dicho número; si, por el contrario, para ese contacto hay almacenado más de un número, al pulsar sobre el icono aparecerá un listado con todos los teléfonos disponibles para que el usuario seleccione a cual desea realizar la llamada, como muestra la Figura 26.

Figura 26: Ventana de números del cliente

Además, para facilitar la búsqueda de clientes, se ha implementado para esta ventana dos posibilidades. La primera consiste en una etiqueta informativa de cuál es la primera letra del elemento del listado que se acaba de situar en la parte superior de la pantalla, como se observa en la Figura 27. Esta implementación se debe a que todos los dispositivos móviles *Android* tienen pantalla táctil y se puede realizar un *scroll* muy rápido en los listados; de esta manera, mientras hace *scroll*, podrá observar en qué parte del listado se encuentra, y no tendrá más que pulsar la pantalla para detenerse en la letra que desee.

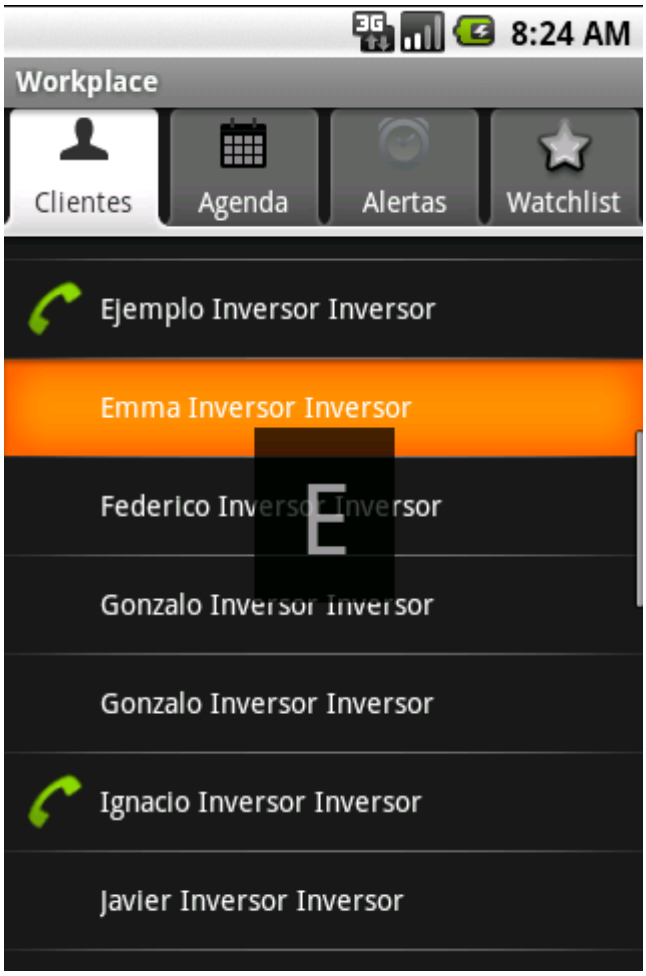


Figura 27: Scroll sobre la ventana de listado de clientes

La otra opción, es pulsar en el botón de búsqueda común a todos los dispositivos *Android* (ver Anexo para más información). En la ventana de búsqueda el usuario debe introducir el patrón que desea buscar y automáticamente le aparecerá un nuevo listado con todos los nombres que contienen dicho patrón de búsqueda. Desde esta nueva ventana se pueden realizar todas las opciones explicadas para esta pantalla, así como pulsar sobre un cliente para acceder al menú con información del cliente. Para volver al listado original simplemente debe pulsarse en el botón volver del dispositivo. En la Figura 28 se muestra el resultado de la búsqueda de clientes que contentan la secuencia “an”.

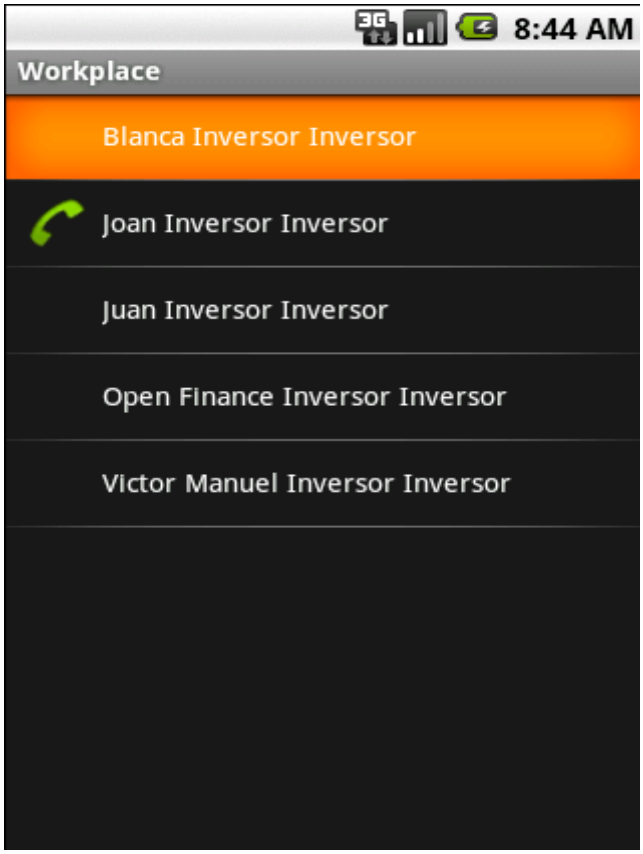


Figura 28: Resultado de la búsqueda

Dentro del menú de usuario, otra de las ventanas que se puede ver es la agenda. Esta pantalla muestra una tabla con las tareas y citas del usuario ordenadas por fecha. En la parte superior se encuentra la cabecera indicando qué información es la que se muestra en cada columna; esta cabecera es fija y, al hacer *scroll*, solo se deslizarán las filas de la tabla. Esta ventana se observa en la Figura 29.

Figura 29: Pantalla de agenda del usuario

Si un elemento contiene algún icono, quiere decir que ocurre alguna acción al pulsar sobre él. Por ejemplo, en la Figura 29 se observa que al final de cada fila hay un icono. En este caso, si el usuario quiere ver más información sobre una cita o tarea no tiene más que pulsar sobre ella. Aparecerá una nueva ventana con toda la información disponible, la cual ha sido implementada siguiendo el estilo de ventana de información extra que sigue *Android*, como podrá observarse si, por ejemplo, en una foto de la galería de imágenes de su dispositivo se pulsa en solicitar información extra. Esta ventana se puede observar en la Figura 30. Debe comentarse que tanto el tamaño como la información a mostrar es adaptativo, ya que la información almacenada en *citas* y *tareas* es distinta. Además, la cantidad de información a mostrar depende del propio elemento en si, puesto que sólo aparecerá los atributos de los cuales tenemos valor almacenado.

Figura 30: Pantalla información extra de agenda

Como se comentaba anteriormente, todas las ventanas del menú de usuario disponen de un menú contextual desde el cual se puede acceder a la ventana de configuración y hacer *logout*. En esta pantalla, a dicho menú se añade la opción de seleccionar sobre qué rango de fechas se desea mostrar la información de la agenda. Por defecto el contenido de esta tabla es las citas y tareas del día actual hasta dos meses en adelante. No obstante, como muestra la Figura 31, este rango puede ser modificado por el usuario mostrando, cada vez que se abre la ventana, las fechas en las que actualmente se está obteniendo la información.

Figura 31: Pantalla modificación intervalo de fecha

Cuando el usuario pulse en la pestaña de Alertas, se mostrará una ventana similar a la descrita anteriormente. En este caso la tabla, como se observa en la figura 32, se divide en alertas de carteras y de productos. De nuevo aparecen en la parte superior dos cabeceras fijas que muestran el nombre del campo que se muestra; la diferencia es que en este caso el usuario puede elegir el campo que desea mostrar en la segunda columna. Para ello no tiene más que pulsar sobre la cabecera y aparecerá en pantalla un listado con todos los campos sobre los que se puede mostrar información, como muestra la Figura 33. En la Figura 32 se observa que la cabecera que puede ser modificada contiene una marca amarilla. Este distintivo será común a todas las tablas de la aplicación cuyas columnas pueden ser modificadas.

Al igual que en la pantalla Agenda, al pulsar sobre una alerta aparecerá la ventana de información extra adaptándose de nuevo a la información almacenada sobre cada alerta. También puede observarse en la Figura 32 que algunas alertas tienen un icono a su izquierda; esto significa que la alarma “ha saltado”. Es decir, que la fecha de activación es anterior al día actual y la fecha de vencimiento posterior al día actual.

Como en el caso anterior, las entradas de la tabla se encuentran ordenadas por fecha (en este caso fecha de activación). No obstante, puesto que en esta pantalla se pueden mostrar diferentes valores, se ha añadido al menú contextual una entrada para que el usuario elija en base a qué campo desea que se ordenen los datos. Esto se realiza a través de un listado similar al de la Figura 33 sólo que, en este caso, en vez de seleccionar el campo a mostrar elegimos el campo por el cual ordenar.

Figura 32: Pantalla alertas del usuario

Figura 33: Cambiar campo a mostrar

Como última pantalla del menú, el usuario puede acceder a su *Watchlist*. En concreto, se muestra el *Watchlist* que el usuario tiene marcado como favorito en su *Workplace*. Podemos observar esta ventana en la Figura 34; al igual que en la pantalla de Alertas, aparece la marca identificadora de que el campo a mostrar puede ser elegido por el usuario. Una diferencia es que en este caso la tabla se compone de tres columnas, de las cuales el usuario puede modificar la información a mostrar de las dos últimas, pulsando sobre la cabecera que quiera modificar. En este caso, pulsando sobre el botón menú, el usuario puede elegir en base a qué campo ordenar la tabla.

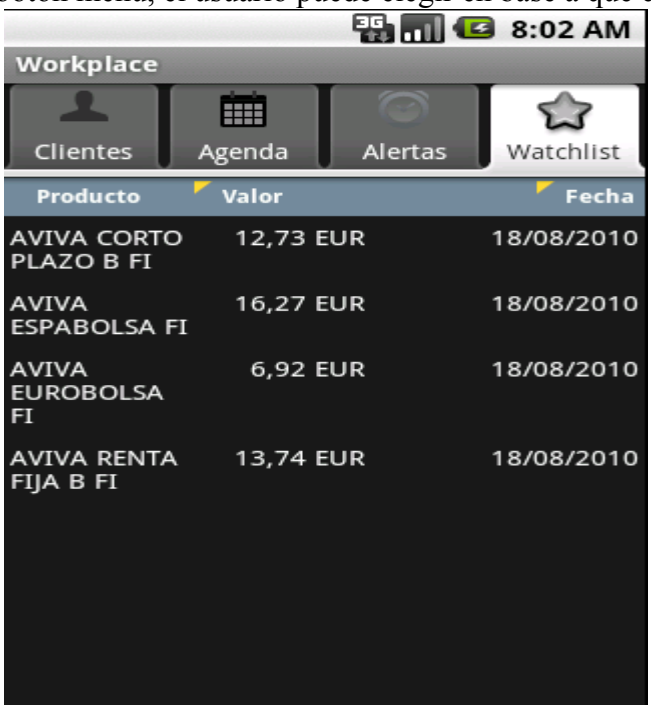


Figura 34: Pantalla watchlist del usuario

Otra diferencia importante de esta pantalla con respecto al resto de las pertenecientes al menú de usuario es que, como en este caso hay bastante información para mostrar y es importante que el usuario pueda ver todo lo que pueda en simultáneo para efectos de comparación, se muestra una columna más si el dispositivo está en posición horizontal. Esto se puede ver en la Figura 35.

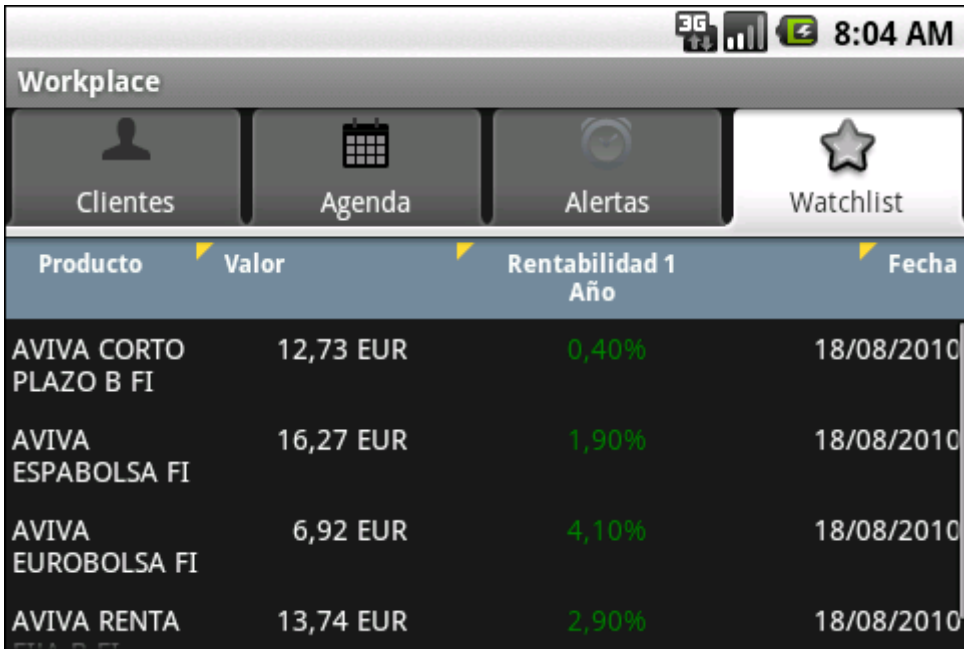


Figura 35: Pantalla watchlist en posición horizontal

6.3 Menú de cliente

A continuación se explican las ventanas que contiene información del cliente seleccionado por el usuario. Para llegar a este menú debe pulsarse en el listado de clientes aquél sobre el que se desee obtener más información. De nuevo este menú se encuentra dividido en cuatro pantallas, que en este caso son: listado de las carteras del cliente, agenda del cliente, alertas del cliente e información del cliente.

Debido a que, en apariencia, la forma de distribuir el menú y el nombre de algunas pestañas coinciden con el menú de usuario, se ha reducido el tamaño de las pestañas, y en la parte superior de la pantalla se muestra el nombre del cliente seleccionado para que el usuario pueda distinguir con facilidad en cuál de los dos menús se encuentra.

La primera pantalla de este menú puede observarse en la Figura 37. En ella se muestra una tabla con información de las carteras de inversión del cliente. Esta tabla está compuesta por tres columnas, de las cuales se puede cambiar la información a mostrar en las cabeceras que tienen el distintivo de esta acción. En este caso, debido a que para una cartera de inversión la cantidad de información almacenada y necesaria para comparar es de suma importancia, al poner el dispositivo en posición horizontal se muestra una columna más, como muestra la Figura 36.

Figura 36: Listado de carteras del cliente en posición horizontal

Figura 37: Pantalla con las carteras del cliente

Se observa también que, en la parte inferior de la pantalla, tenemos una zona de resumen en la cual aparece la suma de los importes de las carteras, la rentabilidad absoluta, y la media de las rentabilidades de todas las carteras. Al igual que las cabeceras de la tabla, esta zona es fija y siempre aparecerá en la parte inferior de la ventana, siendo las filas de la tabla las que se desplazan al hacer *scroll*.

Al igual que las demás ventanas en las que el usuario puede seleccionar qué información quiere mostrar, en este caso, pulsando el botón de menú, se puede acceder a la selección del criterio de ordenación de la tabla, ya que por defecto las carteras se encuentran ordenadas alfabéticamente.

En la Figura 38 podemos ver otra funcionalidad de esta ventana. Se observa que cada fila de la tabla tiene la marca de elemento seleccionable; el usuario puede seleccionar las carteras que desee pulsando sobre ellas para acceder a la pantalla de posiciones de la cartera, la cual se explicará más adelante. Para ello, una vez ha seleccionado una o varias carteras, deberá pulsar en la zona de resumen, la cual también tiene un icono que hace referencia a que se realiza una acción cuando se pulsa en esta zona. En caso de pulsar y no haber seleccionado ninguna cartera, se mostrarán las posiciones de una cartera agregada compuesta por todas las carteras.

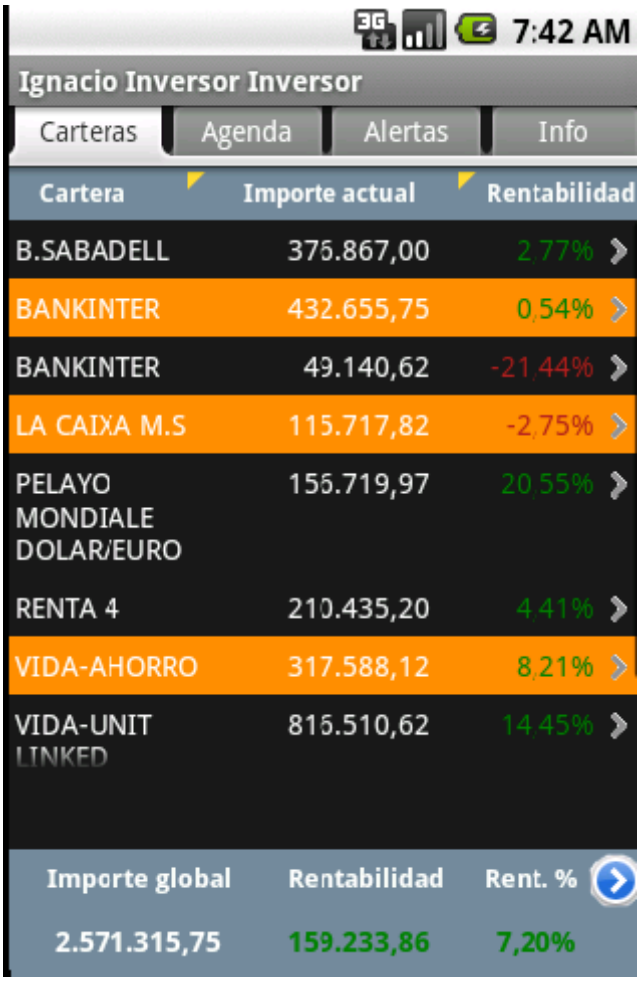


Figura 38: Carteras seleccionadas

Las pantallas de agenda y alertas del cliente son las mismas que las explicadas para el menú de usuario. Tal y como sería de esperar, ofrecen exactamente las mismas funcionalidades, sólo que muestran la información concerniente al cliente. Pueden verse ejemplos en las Figuras 39 y 40.

Figura 39: Agenda del cliente

Figura 40: Alertas del cliente

La última pantalla es la ventana con información del cliente, la cual se muestra en la Figura 41. De nuevo es adaptativa, y el contenido a mostrar depende de la información personal almacenada para cliente en el *Workplace*.

La información está dividida en dos secciones: la primera son los datos identificativos del usuario, los cuales son simplemente informativos, y la segunda es la información de contacto, la cual presenta interactividad. Para cada número disponible podrá realizarse una llamada tan solo pulsando sobre él; si el teléfono fuese de un móvil le dará, además, la opción de mandar un SMS si así se desea, tal y como muestra la Figura 42. En caso de disponer de dirección de correo electrónico, al pulsar sobre él aparece el formulario de la Figura 43 para mandar un email. Tras pulsar el botón enviar, y en caso de que haya más de una aplicación de correo instalada en el dispositivo (*Gmail* siempre por defecto en un *Android*), pedirá al usuario que elija desde cual desea mandarlo. Por último, si disponemos de su dirección, al pulsar sobre ella nos mostrará la ubicación en *Google Maps*, con el icono de la empresa *Openfinance* en el lugar de la localización como muestra la Figura 44.

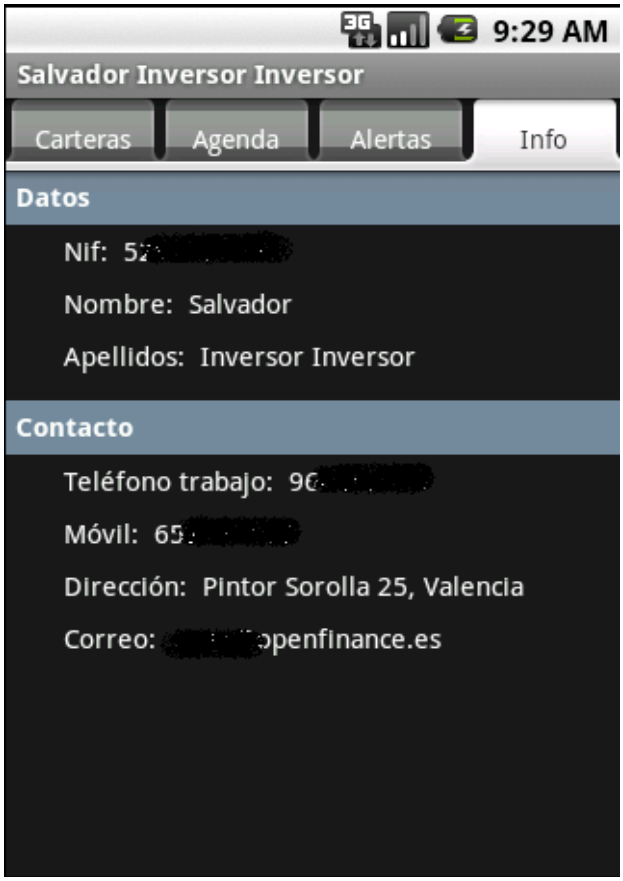


Figura 40: Ventana información del cliente

Figura 41: Formulario para enviar SMS

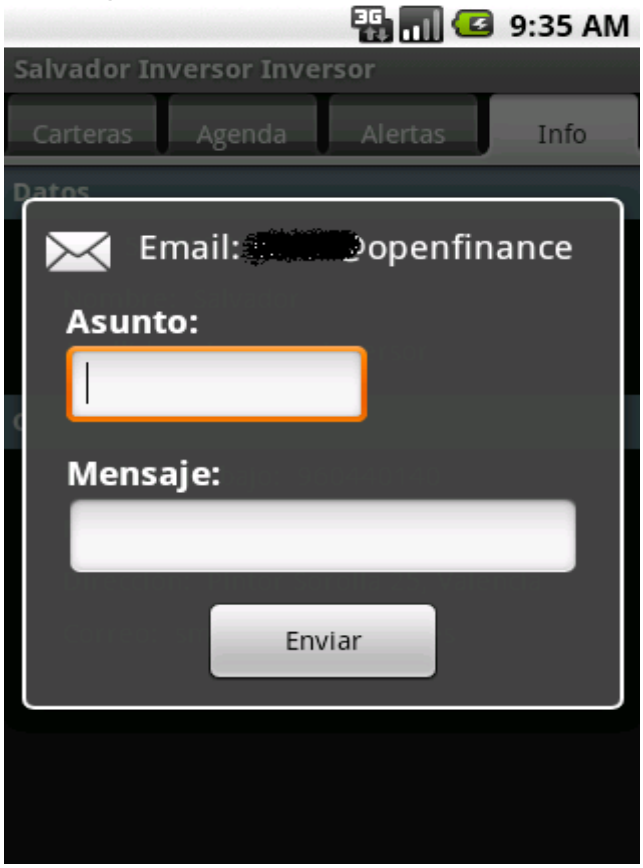


Figura 42: Formulario para enviar email

Figura 43: Ventana GoogleMaps

6.4 Menú de carteras del cliente

Como se comentaba anteriormente, en la pantalla de carteras de inversión del cliente el usuario puede seleccionar una o varias carteras para obtener más información. Tras pulsar en la zona de resumen en la pantalla de las carteras del cliente, llega al menú que se explica a continuación.

En este caso se divide en tres pantallas, a las que se accede con sus correspondientes pestañas. Estas son: posiciones, distribución y evolución. El título de la ventana varía en función de si se ha seleccionado previamente una sola cartera o varias. En el primer caso el título será el nombre del cliente y el nombre de la cartera; en caso contrario aparecerá el nombre del cliente y "cartera agregada".

La primera pantalla de este menú muestra una tabla con las posiciones de las carteras. Esta tabla se divide en acciones y fondos. Como observamos en la Figura 45, es muy similar a la pantalla de carteras del cliente puesto que cuenta con tres columnas, de las cuales en dos puede elegirse el campo a mostrar y una zona de resumen en la parte inferior. De nuevo puede establecerse el criterio de ordenación (por defecto se ordenan alfabéticamente) al pulsar sobre el botón menú, y aparecerá una columna más cuando el dispositivo se encuentra en posición horizontal, como muestra la Figura 46.

Figura 45: Pantalla de posiciones de la cartera

Figura 46: Pantalla de posiciones de la cartera en horizontal

En la Figura 47 se observa la pantalla de distribución de las carteras. En ella aparece un gráfico que representa la distribución en porciones por activos. En la parte inferior aparece una leyenda que muestra ordenados por porcentaje los nombres de los activos junto al tanto por ciento que representan.

Si el usuario prefiere ver la distribución de la cartera en forma de tabla lo puede hacer pulsando el botón "Tabla". El resultado de esta acción es la ventana que muestra la Figura 48, en la cual se encuentra una tabla con el nombre del activo, su importe y valor. En este caso también puede seleccionar si desea ordenar la tabla por nombre o por valor pulsando el botón de menú.

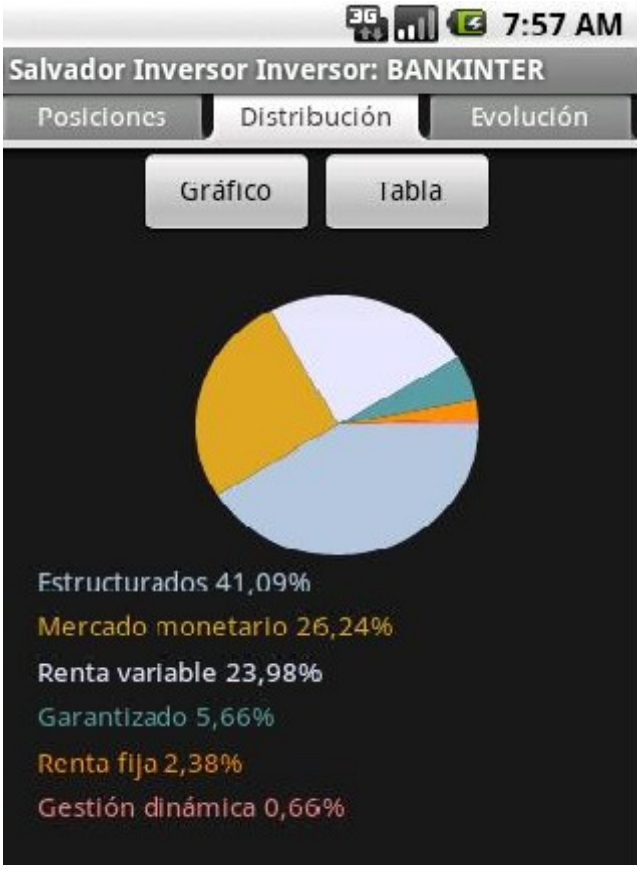


Figura 47: Ventana distribuciones: modo gráfico

Figura 48: Ventana distribuciones: modo tabla

La tercera y última pantalla de este menú es la gráfica que muestra la evolución de la cartera. Como se observa en la Figura 50, los datos a representar por defecto son para el intervalo de un año. No obstante, si el usuario desea ver más detalladamente una zona en concreto, no tiene más que pulsar el botón "cambiar fechas" de la entrada de menú. Esta acción mostrará la misma ventana de selección de fechas que se explicó anteriormente, y que se muestra en la Figura 31.

Antes de explicar los detalles y funcionalidades de esta ventana, en la Figura 49 podemos ver la gráfica de evolución de una cartera que ofrece el *Workplace*, sobre la que se explica en que consiste ésta. Los datos que muestra son los movimientos de la cartera para cada día del intervalo temporal que representa el eje horizontal, con lo cual el primer valor siempre valdrá *100%*, y los sucesivos tomarán su valor en función del valor de sus movimientos respecto a este primer valor.

Figura 49: Gráfica de evolución ofrecida por el Workplace

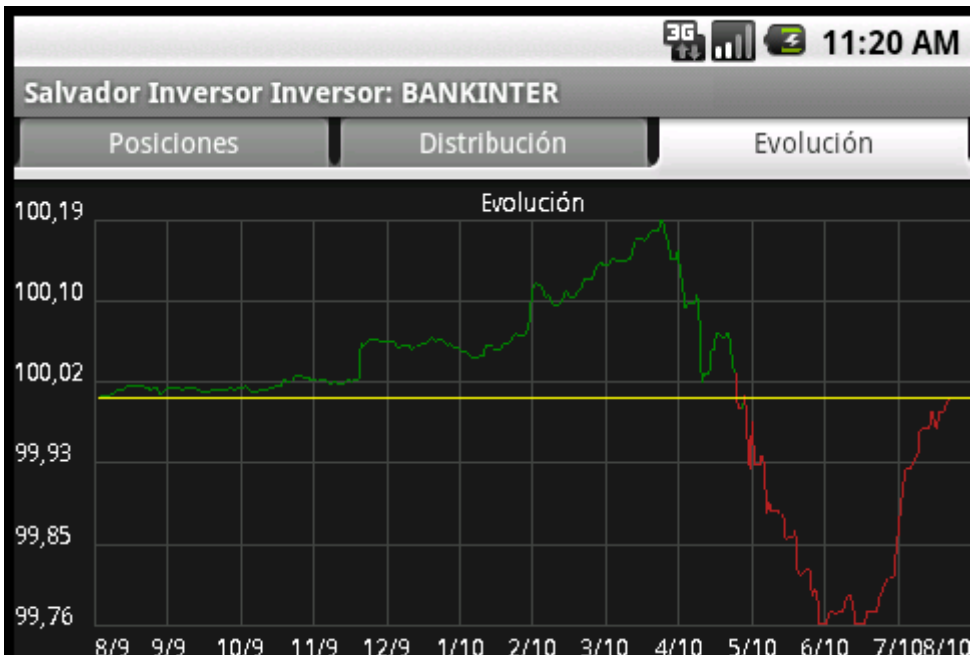


Figura 50: Gráfica de evolución de la aplicación Android

En la Figura 49 se observa la gráfica de evolución obtenida por la aplicación *Android* a partir de los datos devueltos por el servicio web. En este caso la representación es sobre los mismos valores empleados parar dibujar la Figura 50, aunque se ven ligeras diferencias debido a la clase utilizada en este proyecto para representar los datos.

La primera diferencia, la cual en este caso es una ventaja para el usuario, es que en la gráfica de *Android* el eje vertical va desde el valor máximo al valor mínimo de los datos en todos los casos. Esto permite, a primera vista, que el usuario sepa cuál ha sido el valor mínimo y máximo (exactos) en ese intervalo.

Otra diferencia significativa es la línea horizontal que marca la frontera de valores por encima de cien y por debajo, la cual en la aplicación *Android* aporta más información. En este caso los valores iguales o superiores a cien se pintan en color verde, y los inferiores en rojo; esto permite al usuario localizar rápidamente en qué meses se ha producido un descenso o ascenso, y su magnitud. En la imagen se observa que antes de cruzar la frontera del valor cien la línea ya está pintada de rojo; esto se debe a que la línea une valores. Es decir, en este caso de un valor ligeramente superior a cien se pasa a uno ligeramente inferior, lo cual en otras ocasiones permitirá ver al usuario si ha sido un cambio brusco o paulatino, como se ve más detalladamente en la Figura 51. En este caso, si la gráfica no estuviera a dos colores, podría pensarse que la caída a mediados de mayo y la caída a mediados de junio son similares. Pero, con este sistema, en un sólo vistazo se aprecia que una fue repentina y la otra se produjo paulatinamente. Lo mismo podría decirse de las subidas que se produjeron en el primer cuadrante vertical de la gráfica.

La última diferencia es los valores de las etiquetas del eje horizontal. En el caso del dispositivo móvil el mes y año se identifican mediante sus valores numéricos debido al reducido tamaño de su pantalla respecta a la un ordenador personal. No obstante, cuando el intervalo mostrado es igual o menor a siete meses estos se identifican de la misma manera que en el *Workplace*, como se observa en la Figura 51.

Figura 51: Gráfica de evolución para un intervalo de cinco meses

6.5 Menú de configuración

Desde cualquiera pantalla de la aplicación puede accederse a la ventana de configuración, pulsando sobre el botón de menú del dispositivo y seleccionando la entrada configuración. El resultado de esta acción se muestra en la Figura 52.

Figura 52: Pantalla de configuración de la aplicación

Como se aprecia en la imagen, la apariencia de esta ventana es similar a cualquiera de las pantallas de configuración de un dispositivo *Android*. En este caso las opciones se encuentran divididas en configuración sobre el PIN de inicio de la aplicación y la URL de conexión al *Workplace*.

En caso de pulsar en la opción de activar PIN y que no haya ninguno almacenado mostrará un mensaje preguntando si se desea establecer uno en ese momento. En cambio, si se pulsa modificar PIN y no hay ninguno almacenado, mostrará automáticamente la ventana para establecer uno como ya se explicó, y se pude ver en la Figura 22. Si ya hubiese PIN, la ventana a mostrar sería la que muestra la Figura 53. En esta pantalla, una vez se pulse el botón "Guardar" se comprobará que se haya introducido correctamente el PIN actual o no podrá realizarse el cambio; si este fuera correcto se comprueba que el nuevo PIN sea numérico, de cuatro dígitos, y que coincida con el escrito en el campo de confirmación del nuevo PIN.

Respecto a las opciones sobre la URL del servidor, cuando se pulsa sobre conexión segura se muestra un mensaje avisando que esta opción no está implementada actualmente, ya que su finalidad es conectar mediante el protocolo HTTPS, pero actualmente el servidor en el que se realiza la conexión no soporta este tipo de peticiones para el envío de datos cifrados, con lo cual esto es una actualización para el futuro.

Al pulsar en "Servidor de conexión" se muestra una ventana de diálogo con la actual URL, de la que se obtienen los datos como muestra la Figura 54. Debido a que, si la nueva URL es incorrecta, no podrá accederse a ninguno de los datos, cuando el usuario pulsa en "OK" se muestra un mensaje de aviso y confirmación, el cual se detallará más adelante en el apartado de mensajes.

Por si, a pesar de la advertencia, o si el servidor actual estuviera mal escrito o dejase de funcionar, la última opción de la ventana permite volver al servidor al que se conectaba cuando la aplicación fue instalada, mostrando antes de realizar la acción un mensaje de confirmación que se explicará en el siguiente apartado.

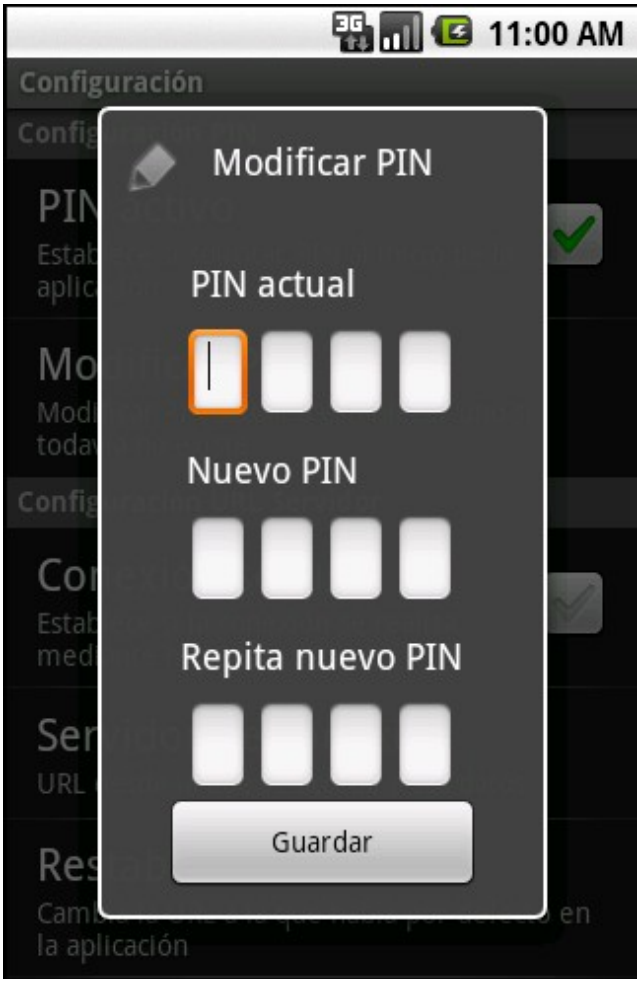


Figura 53: Ventana para modificar PIN

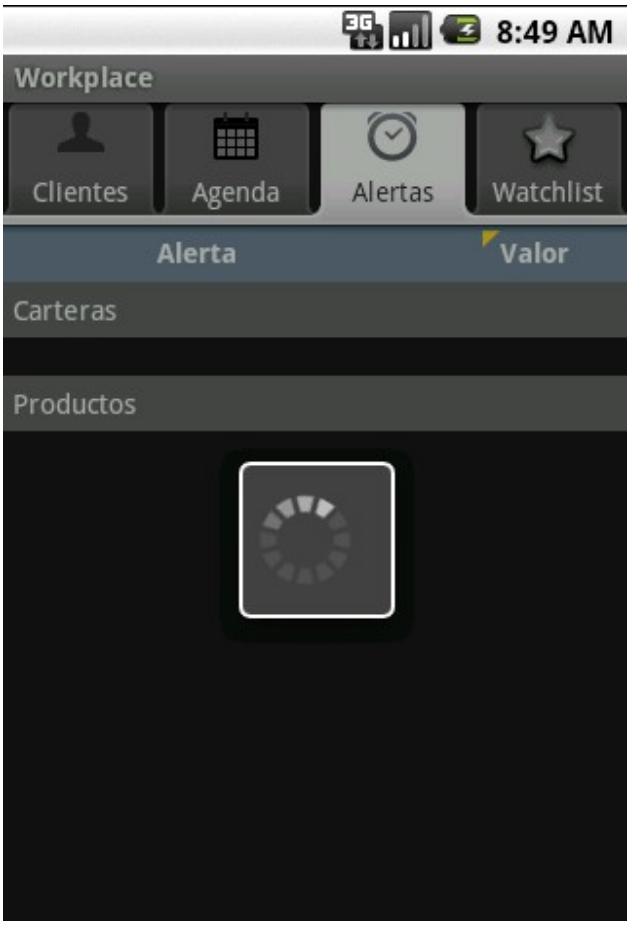
Figura 54: Ventana para modificar URL del servidor

6.6 Mensajes de la aplicación

A continuación se muestran una serie de mensajes que pueden aparecer en cualquiera momento durante la ejecución de la aplicación. El primero y más importante es el mensaje de error de conexión, el cual se observa en la Figura 55. Este aparecerá siempre que se haya perdido la conexión a Internet y se intente realizar una acción que la necesite. Tras pulsar el botón "Aceptar" se cerrará la ventana a la que se intentaba acceder volviendo a la predecesora. En caso de que estemos en el menú de usuario o en la presentación la aplicación, se cerrara ya que no hay ventana con datos correctos a la que volver.

Figura 55: Mensaje de error de conexión

Otro mensaje que puede aparecer en cualquier momento es la pantalla de espera. Ésta aparecerá en las transiciones entre pantallas cuando sea necesario leer datos de Internet y la operación pueda tardar unos segundos (ver Figura 56). Aparecerá una ventana similar pero con mensaje de texto cuando, dentro de una ventana, se vaya a realizar una operación que pueda tardar un tiempo considerable, como por ejemplo cambiar el intervalo de fechas en la ventana de agendas, tal y como se observa en la Figura 57. Ambas bloquean cualquier otra acción que se pudiera realizar para no ralentizar más el proceso.



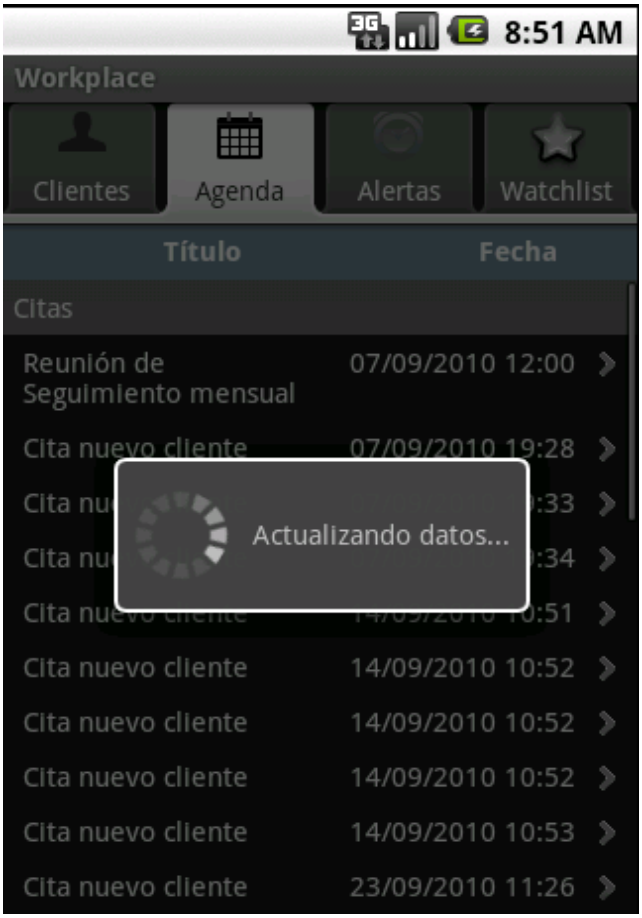


Figura 56: Ventana de espera entre pantallas **Figura 57: Ventana de espera dentro de una pantalla**

A continuación se muestran el resto de ventanas de diálogo que pueden aparecer en la aplicación. Todas ellas muestran un mensaje de aviso en espera de una confirmación o una cancelación, antes las cuales se realizará la acción asociada o no se hará nada.

En la ventana de configuración podemos observar dos mensajes de este tipo. El primero se muestra en la Figura 58 y aparece cuando se solicita modificar la URL del servidor. En este caso si se responde afirmativamente la URL de conexión sera remplazada por la escrita por el usuario, mientras que en caso de respuesta negativa no se hará el cambio. El segundo caso se plantea cuando se solicita restablecer el servidor como se puede ver en la Figura 59, se muestra un mensaje avisando de las consecuencias de realizar la acción. En caso de respuesta afirmativa esta se llevará a cabo y en caso contrario no se hará nada, respectivamente.

Otra opción que se puede dar es que, en vez de respuesta afirmativa o negativa, se plantee realizar una acción u otra. Un ejemplo de este caso se muestra en la Figura 60. Esto ocurre cuando en la ventana de información de un cliente pulsamos sobre su teléfono móvil, y dependiendo de la respuesta del usuario, pasaremos a la pantalla que muestra la Figura 42 (mandar un SMS a un número) o a realizar una llamada telefónica. No obstante, si el usuario no quiere realizar ninguna de las dos acciones, puede cerrar el diálogo pulsando sobre el botón “Volver” de su dispositivo *Android*.

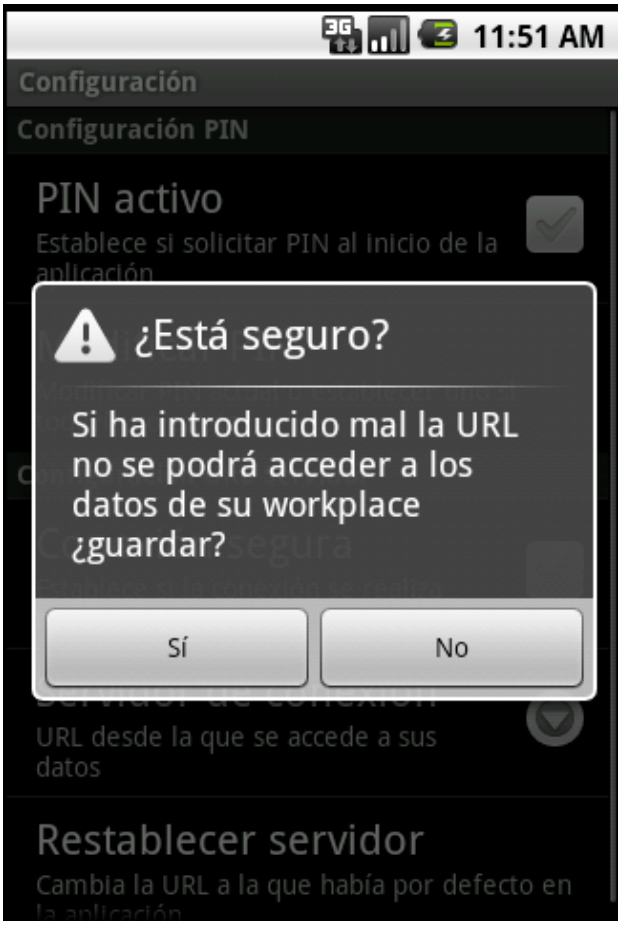


Figura 58: Mensaje confirmación cambio URL

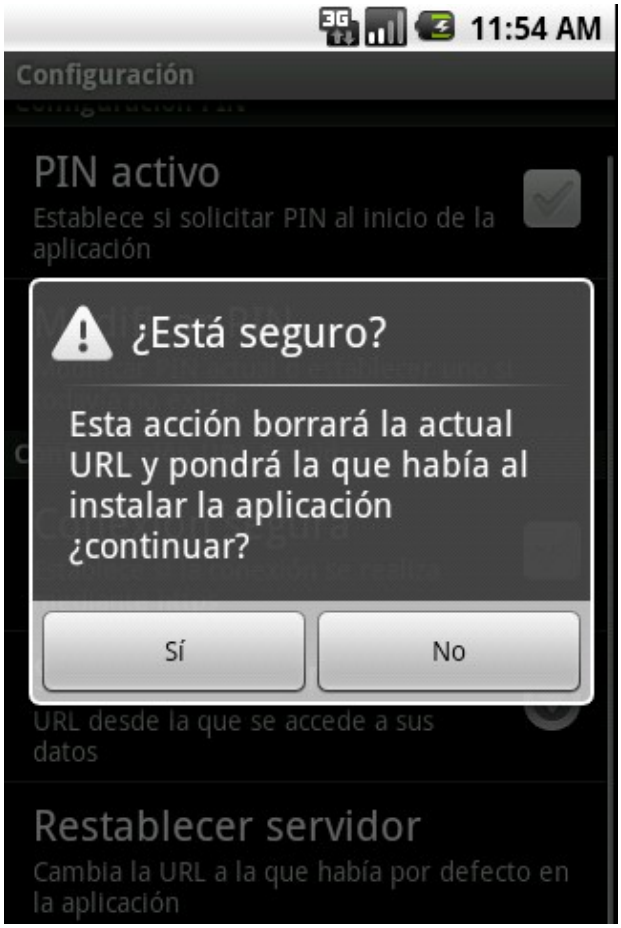


Figura 59: Mensaje advertencia restablecer servidor

Figura 60: Mensaje para escoger acción

7 Ampliaciones

Esta aplicación está preparada para futuras ampliaciones, las cuales pueden ser tanto de mejora de la interfaz como de incorporación de nuevas funcionalidades.

Por ejemplo, hay tres pantallas en las que al girar el dispositivo se muestra una columna más, puesto que en estas ventanas hay mucha información para mostrar y es de gran importancia que unos datos sean comparados con otros. Debido al actual tamaño de pantalla de la mayoría de dispositivos, sólo se muestra una columna más para que toda la información pueda ser visible en cualquier móvil. Sin embargo, el tamaño de las pantallas es cada vez mayor, así que una posible mejora es que sea el propio usuario el que decida cuantas columnas más quiere mostrar al poner su pantalla en posición horizontal.

Para mejoras en el tiempo de carga de los datos, se podría hacer que el intercambio sea mediante notación *JSON*, ya que reduce mucho la cantidad de información no útil a enviar con respecto a *XML*. Por muy avanzados que estén actualmente los dispositivos móviles, sigue siendo crítica su capacidad de procesamiento con respecto a los tiempos de respuesta a los que un usuario está acostumbrado con su *PC*. Para llevar a cabo esto, sería necesario primero tener servicios web que devuelvan los datos en esta notación. El código es altamente independiente de la forma de procesar y mostrar los datos, con lo que sólo sería necesario hacer clases lectoras de *JSON*, además de las ya existentes para *XML*.

Respecto a la seguridad de la conexión, se puede mejorar en dos aspectos básicos: (i) Por un lado, que el servidor permita conexión mediante el protocolo *HTTPS* que, aunque la ventana de configuración lo contempla, activar esta opción actualmente no aporta nada. (ii) Por otro lado, que se pueda establecer el método de autenticación *OAUTH*, dado que como se ha comentado *Basic* es muy básico a nivel de seguridad. Con el uso de este protocolo se evitaría tener que estar enviando usuario y contraseña.

También sería bueno para el usuario que no solo pueda acceder a los datos mediante lectura, sino que también pueda modificarlos o crear nuevos.

8 Conclusión

Todos los objetivos presentados al principio de este documento han sido cumplidos. Tanto los referentes a la aplicación como los objetivos personales.

Con respecto a la aplicación, cualquier usuario del *Workplace* podrá disponer en su *Android* de un programa que le permite un acceso rápido y ordenado a todos sus datos. Además, esta aplicación le aportará varias opciones extra a realizar con los datos de sus clientes, y en ese sentido es una agenda comercial completa con la que podrá localizar y ponerse en contacto con cada cliente de todas las maneras que un dispositivo móvil permite.

Con respecto a la información a mostrar, tiene muchas opciones a realizar con ella, ya que en cada tabla puede seleccionar el campo a mostrar o el campo por el que desea ordenar los datos. Tanto las ventanas de configuración como las de información se han realizado con la misma apariencia a las que el usuario estará acostumbrado en el resto de aplicaciones que *Android* lleva instaladas por defecto.

Referente a los objetivos personales, he aprendido mucho con este proyecto tanto a nivel de programación como en lo referente al mundo laboral. Uno de los objetivos marcados era aprender por mí mismo a trabajar en algo que no hubiera visto en ninguna asignatura. En ese sentido la experiencia ha sido muy positiva, siendo que todos los requisitos marcados por la empresa para la aplicación han sido cumplidos e incluso ha dado tiempo para incorporar funcionalidades que no se pedían, como el envío de SMS o la localización de un cliente en *GoogleMaps*, entre otras. Así como el desarrollo de un servicio web más, puesto que la idea inicial era utilizar sólo los ya existentes.

Antes de realizar este proyecto, había desarrollado aplicaciones en *J2ME* y *Windows Mobile* en asignaturas de la carrera. Recomiendo a cualquier persona que le interese programar para dispositivos móviles que comience por *Android* porque, a pesar de que algunas clases o métodos tienen aún algún pequeño fallo, las posibilidades que esta plataforma ofrece respecto a las anteriores son muy superiores, no sólo a nivel de la API, sino también de recursos a utilizar dados por el propio lenguaje. En el caso de *J2ME*, también su código se escribe en *Java*, pero sin embargo no se tiene acceso a todas las clases y métodos que este lenguaje ofrece; lo mismo ocurre respecto a *Windows Mobile* y *C#*. En cambio, con *Android* se puede complementar todo lo que ofrece su API (que es mucho más que para las otras) con todo lo que existe para *Java*.

Finalmente, cabría decir que no sólo he aprendido a programar para una plataforma más. En realidad, durante la estancia en la empresa, me han ayudado a dar el paso final de ser un estudiante a un Ingeniero Informático, dándome las directrices necesarias para conseguir una correcta organización y planificación, así como realizar un código optimizado y fácil de mantener. Todo lo aprendido en la empresa y con el proyecto son requisitos indispensables para mi futura vida laboral.

9 Bibliografía

ABLESON, FRANK; COLLINS, CHARLIE; SEN, ROBI: *Android: guía para desarroladores*. Madrid: Anaya, 2010.

BLAKE, MEIKE; LOMBARDO JOHN; MEDNICKS, ZIGURD; ROGERS, RICK: *Android application development.* Sevastopol: O´Reilly, 2009.

BURNETTE, ED: *Hello, Android: introducing Google's mobile development platform*. Raleigh: Pragmatic Bookshelf, 2008.

DiMARZIO, JEROME: *Android: a programmer´s guide*. Emeryville: McGrawHill, 2008.

HASHIMI, SAYED; KOMATINENI, SATYA; MACLEAN, DAVE: *Pro Android 2*. Nueva York: Apress, 2010.

MEIER, RETO: *Professional Android 2 Application Development*. Indianapolis: Wyley, 2010.

MURPHY, MARK: *Beginning Android 2*. Nueva York: Apress, 2010.

FORO SOBRE ANDROID [agosto de 2010] disponible en http://www.anddev.org

FORO SOBRE ANDROID [agosto 2010] disponible en http://android-spa.com

PÁGINA OFICIAL DE ANDROID [agosto 2010] disponible en http://developer.android.com

Índice de figuras

Anexos

***Openfinance*:** es una empresa líder en soluciones tecnológicas de asesoramiento financiero y gestión de carteras.

***Workplace*:** en este documento se hace referencia al producto *Workplace* ofrecido por la empresa *Openfinance*. Este producto ofrece la posibilidad de clasificar y segmentar clientes de forma eficaz, realizar propuestas personalizadas, alimentar y mantener una agenda de contactos dinámica, así como llevar a cabo un control exhaustivo y actualizado de la comercialización de productos y servicios. Se ofrece como una aplicación web dónde cada cliente tiene una versión adaptada a sus necesidades específicas.

***EAFIS*:** es una asociación de gestores independientes. Esta asociación tiene una versión del *Workplace* adaptada a sus necesidades concretas.

***Watchlist*:** es una lista en la que el usuario puede añadir las acciones o valores que quiere seguir. De esta forma no tiene que ir buscándolos para ir siguiéndolos, y los puede localizar de una forma rápida.

Botones *Android:* a lo largo del presente documento se han mencionado una serie de botones comunes a todos los dispositivos *Android*. En la imagen pueden verse y a continuación se explica, de izquierda a derecha, cada uno de ellos.

El primer botón, podría denominarse *volver* o botón de retroceso. Sirve para regresar a la pantalla anterior, o para salir de la aplicación en el caso de que se encuentre en el menú principal.

El siguiente, es el que durante este documento se nombraba como *botón de menú* ya que su única función es mostrar en la parte inferior de la pantalla el menú contextual asociado a la ventana que está visible en ese momento.

El tercer botón sirve para volver a la pantalla principal del teléfono sin cerrar la aplicación, la siguiente vez que se entre a la aplicación regresará automáticamente a la pantalla en la que se estaba cuando se pulso este botón.

El último botón es el de búsqueda. Una vez es pulsado aparece en la parte superior de la pantalla un cuadro para introducir el texto referente a lo que se desea buscar. Cuando la ventana en curso no tiene activada la opción de búsqueda ésta se realiza sobre Internet. En el caso de esta aplicación, la búsqueda sólo está activada para la pantalla del listado de clientes.