***Título**: Aplicación Android para Supermercados*
***Autor**: Rubén García Padilla*

***Fecha**: XX/06/2011*

***Director**: Lluís Pérez Vidal*
***Departamento del director**: LSI*

***Titulación**: Ingeniería Informática Técnica de Gestión*

***Centro**: Facultat d'Informàtica de Barcelona (FIB)*
***Universidad**: Universitat Politècnica de Catalunya (UPC)*

Índice

1. Introducción

Este capítulo describe el entorno en el que se ha desarrollado mi proyecto final de carrera, cuales son sus objetivos y la motivación por la cual he decidido crear la aplicación. Además explicaré una breve descripción del contenido de cada capítulo.

1.1. Contexto y objetivos del proyecto

El proyecto se ha llevado a cabo utilizando el lenguaje de programación Android. Se ha decidido usar dicho lenguaje ya que lo que pretendía era hacer una aplicación portable y con la cual una gran cantidad de usuarios puedan compartir ideas. Además es un software libre el cual ofrece muchas facilidades a la hora de desarrollar código.

Esta aplicación, llamada *EasyMarket*, tiene como objetivo simular una compra por parte del usuario de manera que el comprador obtenga una propuesta que se ajuste lo máximo posible a sus preferencias según ciertos criterios. Además de esto ofrece una amplia gestión de productos y centros los cuales él podrá ampliar si es necesario.

El usuario podrá comentar, valorar, actualizar todos los tipos de productos y supermercados que existen con tal de compartir entre todas las personas sus criterios para lograr así una amplia diversidad de opiniones para consultar. Más adelante explicaremos con más precisión los servicios que ofrece este programa.

Sin embargo, para llegar a la esencia del programa, debemos cumplir unos objetivos previos:

- **Conocimiento del lenguaje de programación Android y su entorno de desarrollo:** Puesto que mis conocimientos en este ámbito eran prácticamente escasos, he tenido que investigar y ampliar mi conocimiento sobre este SO.

- **Estudio de los diversos supermercados más extendidos:** Para poder tener una aproximación lo más exacta posible de qué centros son los más populares he realizado una búsqueda exhaustiva de ellos y los productos que pueden poseer.
- **Definición de los límites de la aplicación:** Para tener una definición exacta de todas las funcionalidades de la aplicación, previamente se han tenido que detallar lo mejor posible para evitar ambigüedades que puedan retrasar el momento de entrega.
- **Diseño de la aplicación:** Una vez pensados los objetivos, es hora de crear una estructura y un diagrama que pueda relacionar todas las clases necesarias.
- **Depuración y pruebas:** Como sucede en toda parte final de desarrollo de una aplicación, el testeo y la optimización del código es algo esencial para que todo funcione de la manera más correcta posible.

1.2. Motivación Personal

Lo que me ha motivado a realizar este proyecto tiene diversas razones.

- **Nuevo lenguaje, nuevos objetivos.** El aprendizaje personal de desarrollar una aplicación dentro de un lenguaje que no conocía me suponía un nuevo reto a cumplir.
- **Aplicación móvil, comodidad asegurada.** Dado que los móviles últimamente son herramientas muy versátiles, creía original añadir este tipo de programa ya que no conocía ninguno para esta plataforma que ofreciese tales servicios.
- **Conocimiento aprendido en práctica.** Puesto que proceso completo de creación de una aplicación conlleva tocar un gran abanico de conocimientos, me sentía estimulante intentar poner en práctica casi todo lo aprendido a lo largo de la carrera para también, de alguna manera, demostrarme a mí mismo qué he aprendido en todos estos años.

- **Una mirada al futuro.** Es posible que, si esta aplicación tiene bastantes críticas positivas, en un futuro pueda ampliarse añadiendo nuevas acciones o tener éxito. Todo esto visto desde un punto optimista.

1.3. Capítulos de la Memoria

En este apartado se resume brevemente el contenido de los capítulos que componen la memoria.

En el **capítulo 2**, se explica la situación actual de este SO[1] y veremos qué expectativas tiene de cara al futuro. Además, veremos su diseño y características a la hora de programar Android.

El **capítulo 3**, muestra las tecnologías que he usado a la hora de implementar toda la aplicación. Antes de mostrar las funcionalidades del sistema es imprescindible ver qué herramientas he usado y explicar de qué se compone un programa Android.

En el **capítulo 4**, mostraremos todo el diseño de la aplicación. También explicaremos sus características principales y que operaciones he llevado a cabo para mostrarlas. Se explicará la implementación de dos operaciones más complejas para justificar dichos resultados. Con todo esto se comprenderá finalmente como funciona al completo la aplicación.

En el siguiente **capítulo 5** presentaremos la planificación del proyecto y los costes que ha supuesto.

Veremos que conclusiones he sacado finalmente de este proyecto en el **capítulo 6**.

En el **capítulo 7** están listados todos los lugares que he necesitado acceder para mi formación y consulta sobre Android y temas relacionados.

El **capítulo 8** se muestra los agradecimientos.

[1] Siglas de Sistema Operativo

2. Fundamentos de Android

En este capítulo vamos a explicar a grandes rasgos ciertos conocimientos acerca de Android.

Veremos que es Android en su definición más estricta. También comprenderemos como esta diseñado y qué características tiene.

2.1. Definición y evolución de Android

Android es un sistema operativo móvil basado en Linux[2], que junto con aplicaciones middleware está enfocado para ser utilizado en dispositivos móviles como *smartphones*, *tablets*, etc. Es desarrollado por la Open Handset Alliance[3] la cual es liderada por Google.

Las unidades vendidas de smartphones con Android han ido evolucionando hasta colocarse en el 2010 en el primer puesto en EEUU[4].

A nivel mundial alcanzó una cuota de mercado del 50,9% durante el cuarto trimestre del 2011, mas del doble que el segundo sistema operativo (*iOS Phone*) con más cuota[5].

En la siguiente estadística de la *Figura 2.1* podremos ver con más exactitud el uso de los diversos SO que existen:

[2] Sistema Operativo de software libre.
[3] Es un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio.
[4] Según http://blog.nielsen.com/nielsenwire/online_mobile/android-most-popular-operating-system-in-u-s-among-recent-smartphone-buyers/
[5] Según http://www.gartner.com/it/page.jsp?id=1924314

Top Smartphone Platforms 3 Month Avg. Ending Mar. 2012 vs. 3 Month Avg. Ending Dec. 2011 Total U.S. Smartphone Subscribers Ages 13+ Source: comScore MobiLens			
	Share (%) of Smartphone Subscribers		
	Dec-11	Mar-12	Point Change
Total Smartphone Subscribers	*100.0%*	*100.0%*	*N/A*
Google	47.3%	51.0%	3.7
Apple	29.6%	30.7%	1.1
RIM	16.0%	12.3%	-3.7
Microsoft	4.7%	3.9%	-0.8
Symbian	1.4%	1.4%	0.0

Figura 2.1 – Uso de plataformas móvil

El sistema operativo móvil Android, usado en smartphones y tablets y desarrollado por Google, ha sido un éxito. Las cifras invitan al optimismo, Google suma más de 850.000 activaciones de dispositivos al día. Se prevé que esta proliferación siga aumentando a lo largo de este año[6].

Finalmente es muy importante decir que tiene una gran comunidad de desarrolladores escribiendo aplicaciones para extender la funcionalidad de los dispositivos. A la fecha, se han sobrepasado las 400.000 aplicaciones (de las cuales, dos tercios son gratuitas) disponibles para la tienda oficial de aplicaciones de Android: *Google Play*, sin tener en cuenta aplicaciones de otras tiendas no oficiales. Los programas están escritos en lenguaje Java con *Android Software Development Kit* (Android SDK). No obstante, no es un sistema operativo libre de malware aunque la mayoría de ello es descargado de sitios de terceros[7].

[6] Según http://tecnologia.elpais.com/tecnologia/2012/02/27/actualidad/1330361316_288568.html
[7] Según http://www.securitybydefault.com/2011/02/evolucion-del-malware-en-dispositivos.html

2.2. Características principales de Android

Los componentes principales del sistema operativo de Android (cada sección se describe en detalle):

- **Aplicaciones**: las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.
- **Marco de trabajo de aplicaciones**: los desarrolladores tienen acceso completo a las mismas *Apis* del *framework* usadas por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes. Cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del *framework*). Este mismo mecanismo permite que los componentes sean remplazados por el usuario.
- **Bibliotecas**: Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android. Algunas son: *System C Library* (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y *SQLite*, entre otras.
- **Runtime de Android**: Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual *Dalvik*. *Dalvik* ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. *Dalvik* ejecuta archivos en el formato *Dalvik Executable* (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros y corre clases compiladas por el compilador de Java que se transformaron al formato .dex por la herramienta incluida "dx".
- **Núcleo Linux**: Android depende de Linux para los servicios base del

sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

La estructura de Android esta formada por varias capas: Kernel de Linux, Librerías, Frameworks y aplicaciones.

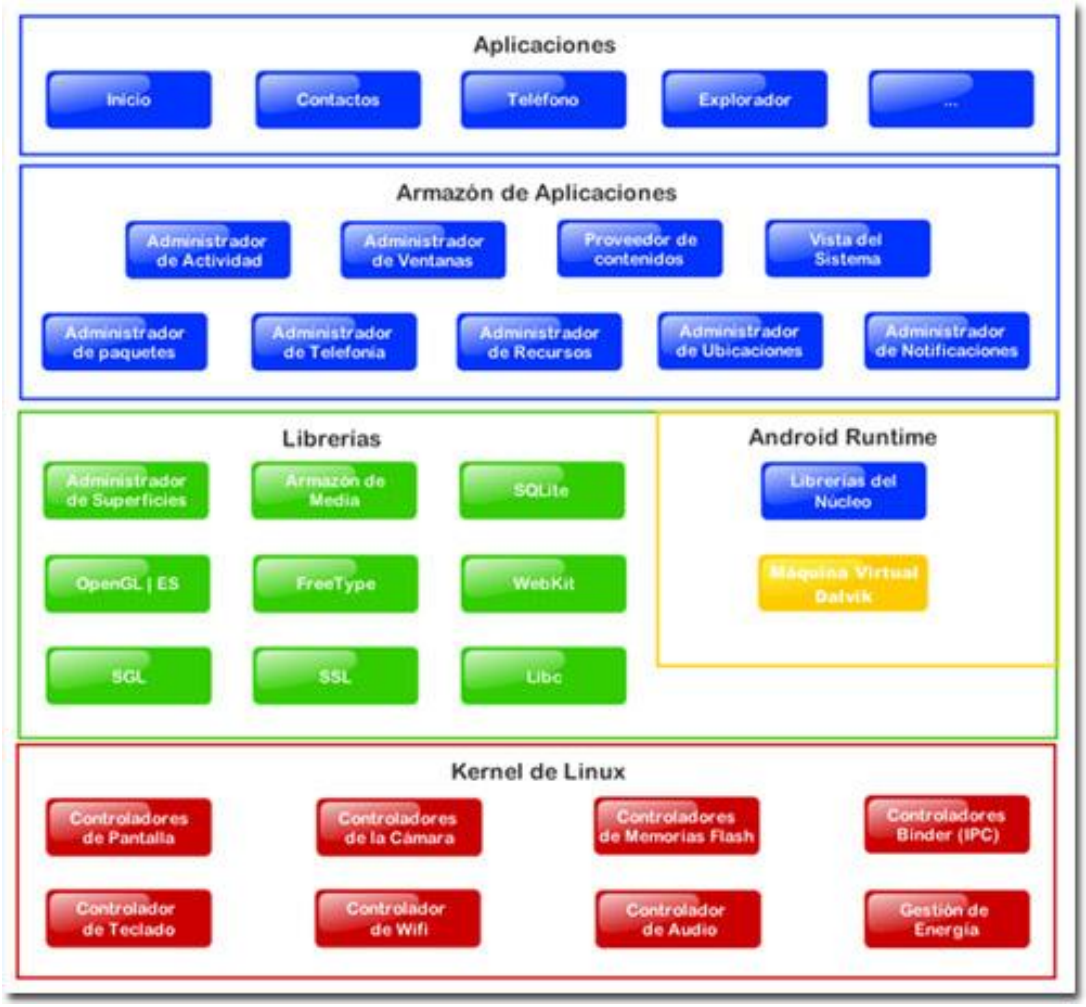


Figura 2.2 – Arquitectura Android

3. Desarrollo y herramientas

Para llevar a cabo el desarrollo de *EasyMarket* he tenido que utilizar ciertas tecnologías. Comentaré brevemente las más básicas para crear una aplicación y explicaré de maneras justificadas aquellas que son herramientas externas o que he creído más interesantes profundizar un poco más.

Tecnologías básicas:

- **Eclipse Classic[8]:** es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar aplicaciones.
- **Java Development Kit[9]:** se puede definir como un conjunto de herramientas, utilidades, documentación y ejemplos para desarrollar todo tipo de aplicaciones Java.
- **ADT Plugin[10]:** es un *plugin* para el IDE (Entorno de Desarrollo Integrado) de Eclipse que esta diseñado para crear un entorno potente e integrado con el fin de crear aplicaciones Android.
- **SDK de Android[11]:** es un kit de desarrollo necesario para crear herramientas Android. En él podemos encontrar entornos de desarrollo, librerías, emuladores de móviles, documentación, tutoriales, códigos de ejemplo, etc.

Una vez instalado y configurado este entorno ya podemos pasar a desarrollar aplicaciones Android en nuestro ordenador. La plataforma para instalarlo puede ser GNU/Linux, OSX o Windows.

[8] Se puede obtener desde http://www.eclipse.org
[9] Se puede obtener desde http://www.oracle.com/technetwork/java/javase/downloads/jdk6-jsp-136632.html
[10] Se puede obtener desde http://eddfox.blogspot.com.es/2011/09/como-instalar-adt-plugin-para-eclipse.html
[11] Se puede obtener desde http://developer.android.com/sdk/index.html

3.1. Activities y XMLs

Para empezar a desarrollar una aplicación debemos entender que es un *Activity*. Un *Activity* es una clase pública en la que el usuario puede desarrollar el código que desee. Este tipo de clases vienen heredadas de la clase base "*android.app.Activity*".

Habitualmente las aplicaciones se componen de activities. Cuando se solicita un *Activity*, esta pasa al primer plano colocándose encima de otras, formado así una pila de activities. El botón del dispositivo "Back" cierra el *Activity* actual y recupera el primero de la pila.

En Android una aplicación no tiene control de ciclo de vida por lo que debemos poder finalizarlas en cualquier momento. Cada programa ejecuta su propio proceso. El *runtime*[12] de Android se encarga de gestionar una aplicación, y por tanto las activities que la forman.

La actividad o *Activity* representa una acción que puede realizar el usuario. Se corresponden en muchos casos con al UI (Interfaz de usuario). Muestran los *widgets*[13] dispuestos y gestiona las interacciones del usuario con estos. Toda actividad debe estar declarada en el AndroidManifest.xml (es un fichero el cual se deben declarar todos activities y todos aquellos permisos adicionales (Internet, Cámara, etc.) que el sistema necesite para que este lo lea antes de ser ejecutada).

[12] Intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo.
[13] pequeñas aplicaciones informáticas en forma de iconos gráficos que se utilizan en páginas web para permitir la interactuación con el usuario o visitante.

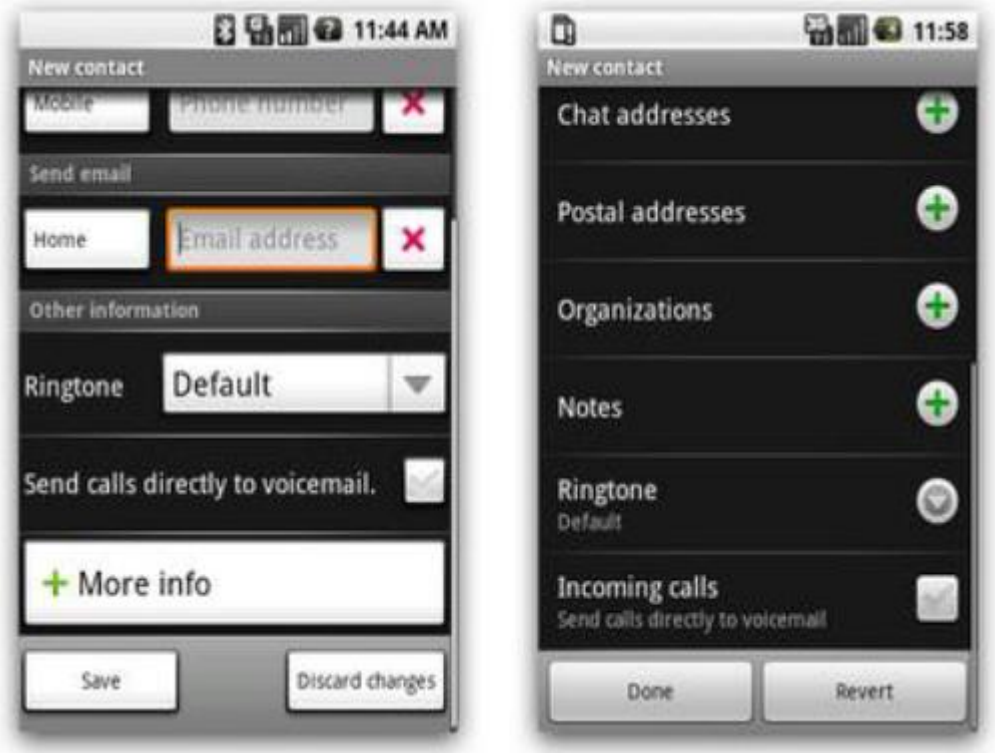


Figura 3.1 – Plantillas XMLs

Para comunicar activities entre ellos Android nos proporciona ciertos métodos. Para lanzarlos primero debemos declararlos dentro de un *Intent* que es una clase en la que permite especificarlo.

- **startActivity(intent):** lanza el *Activity* sin esperar resultados.
- **startActivityForResult(intent, requestCode):** lanza el *Activity* de la cual esperamos un resultado. Cuando el *Activity* llamado finaliza, se invoca el método onActivityResult() pasándole el *requestCode* con el que se lanzó el *Activity*.

Para crear un *Activity* seria de la siguiente manera:

```
public class HelloActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.screen);

   }
}
```

- **onCreate:** llamamos a ese método al crear el *Activity.*
- **setContentView:** asigna a la vista el contenido del layout[14].
- **R.layout.screen:** definición de la vista de la aplicación.
- **Bundle savedInstanceState:** contiene el estado de la actividad anterior que ha sido suspendida.

Durante la vida de un *Activity* esta pasa por una serie de estados. La clase Activity nos permite redefinir ciertos métodos mediante *@Override*, en sus clases derivadas que incluyen el código a ejecutar en las transiciones de estados.

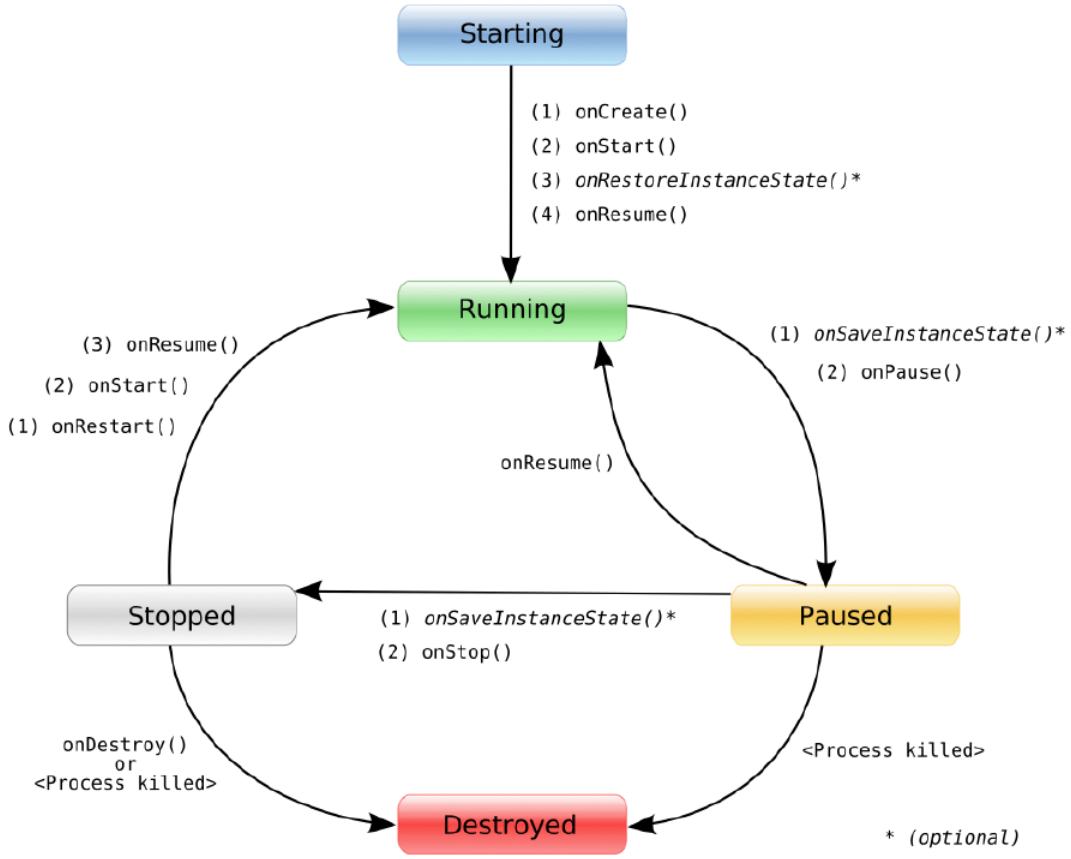


Figura 3.2 – Estados de un Activity

[14] Pagina de diseño.

Estados de un *Activity*:

- **Activo (Running):** el *Activity* esta en la cima de la pila, es visible y tiene el foco.
- **Pausado (Paused):** el *Activity* es visible pero no tiene el foco. Llegamos a este estado lanzando activities transparentes o que no ocupan toda la pantalla.
- **Parado (Stopped):** cuando el *Activity* no es visible. Se recomienda guardar el estado de la UI.
- **Destruido (Destroyed):** cuando el *Activity* termina, o es finalizada por Android. Sale de la pila.

Métodos de transición de estados:

- **onCreate(Bundle):** se invoca cuando el *Activity* arranca por primera vez. Se utiliza para tareas de inicialización. Su parámetro es "*null*" o bien un estado guardado con anterioridad.
- **onStart():** se invoca cuando un *Activity* va a ser mostrado al usuario.
- **onResume():** se invoca cuando un *Activity* pausado va a empezar a interactuar con el usuario.
- **onPause():** se invoca cuando el *Activity* va a pasar al fondo porque otro *Activity* ha sido lanzado para ponerse en primer plano. Se utiliza para conservar el estado del *Activity*.
- **onStop():** se invoca cuando el *Activity* va a dejar de ser visible y no va a ser usada durante un periodo de tiempo largo. Si hay escasez de recursos en el sistema, este método podría llegar a ser ignorado destruyendo el *Activity* directamente.
- **onRestart():** se invoca cuando un *Activity* parado pasa a estar activo.
- **onDestroy():** se invoca cuando un *Activity* va a ser destruido. De igual forma que con onStop(), en caso de escasez de recursos podría llegar a ser ignorado, destruyendo el *Activity* directamente.
- **onSaveInstanceState(Bundle):** se invoca para permitir al *Activity* guardar su estado. Normalmente no necesita ser redefinido.

- **onRestoreInstaceState(Bundle):** Se invoca para recuperar el estado guardado por onSaveInstaceState(). Normalmente no necesita ser redefinido.

Las *layouts* en XML's son aquellos ficheros llamados después de la creación de la clase para mostrar la interfaz de usuario creada. Cada pantalla se implemente como un *Activity* diferente. Estos diseños se guardan en la carpeta "*res/layout*" del programa y para llamarlos se utiliza la clase R (es una clase autogenerada al compilar el código que reúne todos los identificadores de los widgets declarados). La función que se ejecuta con *setContentView(R.layout.pantalla).*

3.2. Almacenamiento de datos

La aplicación EasyMarket contiene una gran capacidad de datos ya que debe gestionar muchos tipos de productos, centros, localizaciones, etc. Para ello he necesitado tener una base de datos en el servidor y otro en la aplicación. A partir de ahí, con las dos bases de datos creadas en cada lugar, he creado una aplicación Java desde el servidor que conecta con el cliente para transferir datos entre ellos.

Para empezar vamos a explicar de qué manera he gestionado el almacenamiento de datos en Android. A pesar de que este SO provee de varios métodos para guardar datos persistentes de tu aplicación, he decidido utilizar el soporte que ofrece esta plataforma sobre una base de datos llamada *SQLite*. Esta librería es *OpenSource*[15] y proporciona capacidades de una base de datos relacional.

Me he decidido a usar *SQLite* y no otros métodos que ofrece Android porque es una base de datos muy extendida en el mundo y otras plataformas, como por ejemplo OSX, también la utilizan en sus dispositivos móviles. Además he creído conveniente compartir las mismas estructuras de datos tanto en el servidor como en el cliente.

[15] Programación Libre (gratuita)

Esta BD (Base de Datos) se almacena en la carpeta del proyecto "*/data/data/<package-name>/databases*". Por defecto son privadas a la aplicación pero por sentencias SQL se puede acceder a ella. Para compartir datos entre aplicaciones con este otras aplicaciones se usan los *Content Providers* (clases que permiten comunicar datos).

Para acceder con estas sentencias SQL a la base de datos trabajamos con cursores. Estos cursores es muy recomendables cerrarlos (con la sentencia *close()*) una vez realizada la consulta para liberar los recursos. Si en acciones próximas decidimos lanzar otra consulta con el mismo cursor solo tendremos que llamar a la función *requery()* actualizando el contenido. Android nos da el método *managedQuery(…)* el cual gestiona el ciclo de vida del cursor por nosotros asociándolo al ciclo de vida de la *Activity*. Llamar a este método es la manera más fácil de usar cursores desde un *Activity*.

En la siguiente imagen (*Figura 3.3*) veremos algunas de las acciones para crear una base de datos, crear/borrar una tabla e insertar/actualizar/borrar una fila. En ella podemos observar como las sentencias son puramente SQL y se pueden usar todo tipo de condicionales.

```
public class Database extends SQLiteOpenHelper {
    private String nombre_centro = "CENTRO";
    private float valoracion = (float) 3.5;
    private int ncentros = 0;


    //Creacion de la BBDD
    public Database(Context context) {
        super(context, "NEW_DATABASE", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db){
        //Creacion de la tabla
        db.execSQL("CREATE TABLE centros (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "nombre_centro TEXT UNIQUE, " +
                "valoracion FLOAT, " +
                "nproductos INTEGER);");

        //Insertar fila
        db.execSQL("INSERT INTO centros (nombre_centro, valoracion, nproductos) " +
                "VALUES ('"+nombre_centro+"', "+valoracion+", "+ncentros+")");

        //Actualizar fila
        db.execSQL("UPDATE centros SET ncentros = "+1+" " +
                "WHERE nombre_centro = '"+nombre_centro+"'");

        //Borrar fila
        db.execSQL("DELETE FROM centros WHERE nombre_centro = '"+nombre_centro+"'");

        //Borrar tabla
        db.execSQL("DROP TABLE centros");

    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

}
```

Figura 3.3 – Sentencias SQL en Android

Sin embargo para insertar o actualizar una fila de la tabla existe otra manera de hacerlo. Para usar esta manera se crea un objeto de la clase *ContentValue* en el cual se le meten los atributos a tratar y se llama a las funciones estándar *insert* o *update* que tiene la clase *Database* de Android. Esta clase también contiene la función *delete* para eliminar filas.

Para configurar la base de datos en mi servidor he necesitado usar la herramienta de *MySQL Server*[16]. Este es un sistema de administración de bases de datos (*Database Management System, DBMS*) para bases de datos relacionales. En él tengo el registro de todos los objetos que vienen por defecto en la aplicación para la instalación. Además esta guarda los nuevos comentarios y

[16] Se obtiene desde http://dev.mysql.com

valoraciones que la gente vaya aportando desde su aplicación en el momento que la actualice.

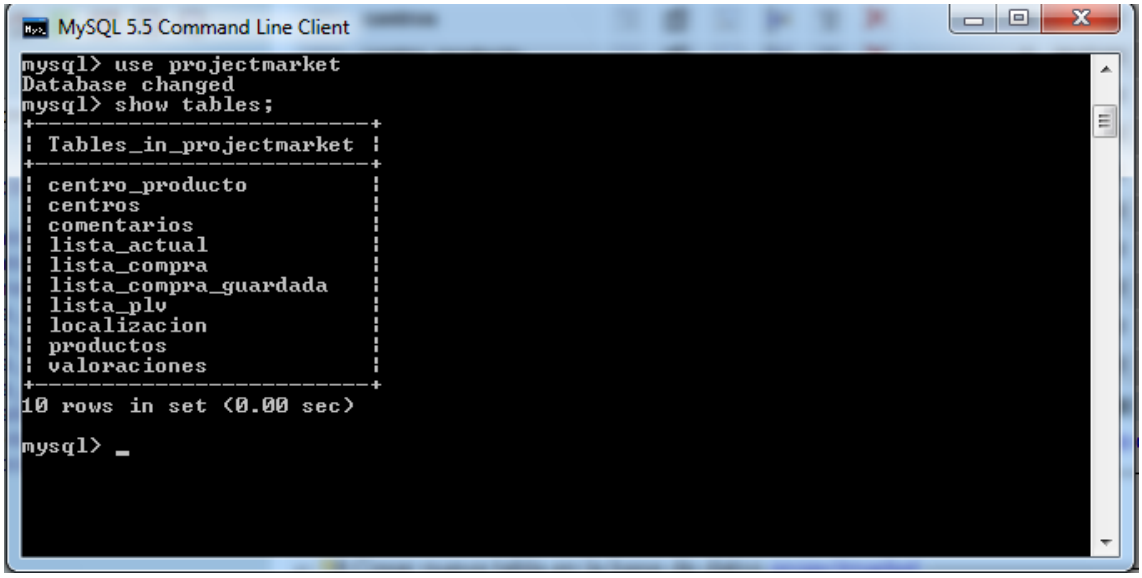


Figura 3.4 – Base de datos de MySql

3.3. Conexiones vía socket con Java

Una vez creada las dos bases de datos lo que queremos es que se comuniquen entre ellas para que se actualice los datos del servidor y el cliente. Para ello he creado una aplicación Java en el servidor que se encarga de recibir peticiones de conexión (*multithread*[17]) y de esa manera abrir un socket entre el cliente y él para establecer la transferencia de datos.

La estructura del funcionamiento por sockets multithread se muestra en la siguiente imagen (*Figura 3.5*):

[17] Con varios procesos en paralelo.

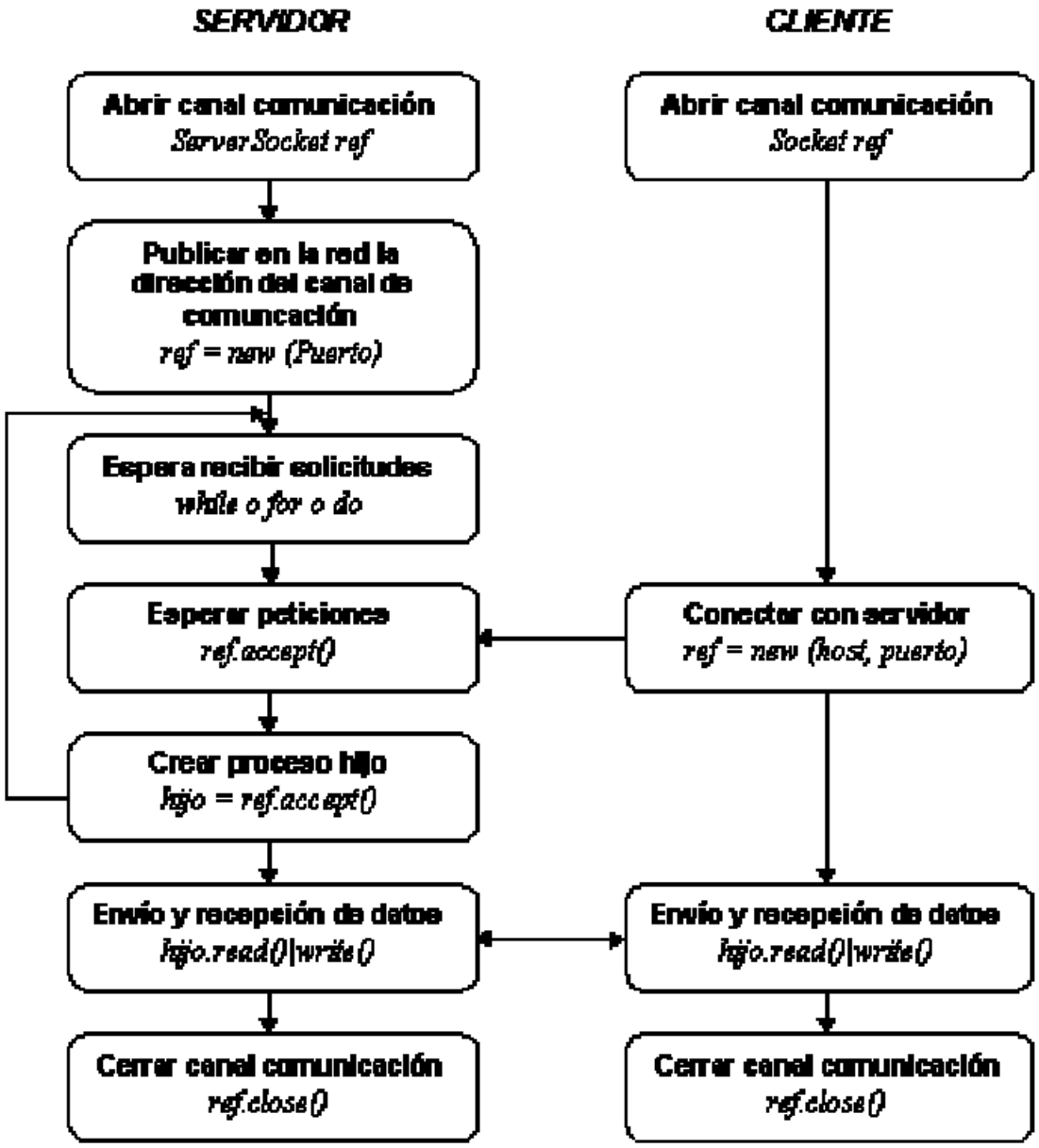


Figura 3.5 – Estados Servidor-Cliente con Sockets

Además, para poder tener una conexión directa con la base de datos MySQL desde la aplicación del servidor he tenido que bajarme un driver que me permita conectarme desde el programa de Java a la BBDD con mi usuario y mi contraseña. El conector utilizado se llama *MySQLConnector*[18].

Una vez añadido el conector al paquete del proyecto del servidor lo que haremos es establecer la conexión con el cliente.

[18] Se obtiene desde http://www.mysql.com/products/connector/

En la *Figura 3.6* veremos como el servidor envía paquetes de datos al cliente. Cabe decir que el siguiente código es lo que el servidor hace únicamente con un cliente. Previamente la aplicación tiene un bucle el cual espera peticiones del cliente (tipo *Runnable*) y las ejecuta mediante un *Thread* nuevo enviándolas aquí.

Servidor enviando productos:

```
String url = "jdbc:mysql://localhost/projectmarket";
Connection con = null;
Statement stmt = null;
ArrayList<Producto> listaP = new ArrayList<Producto>();
//PASO 1 - Activando JDBC
try {
    Class.forName("com.mysql.jdbc.Driver");
}
catch(java.lang.ClassNotFoundException err){
    System.out.println("Clase no encontrada: "+err);
}
//PASO 2 - Conectando con la Base de Datos 'projectmarket'
try {
    con = DriverManager.getConnection(url, "root", "rugapa");
    //PASO 3 - Creamos estado de la conexion con la BBDD
    stmt = con.createStatement();
    //PASO 4 - Creamos una consulta y la enviamos
    String sentenciaSQL = "SELECT * FROM productos";
    ResultSet rs = stmt.executeQuery(sentenciaSQL);
    //PASO 5 - Obtenemos el resultado de la sentencia
    while(rs.next()){
        Producto p = new Producto();
        p.id =rs.getInt("_id");
        p.producto = rs.getString("producto");
        p.ncentros = rs.getInt("ncentros");
        p.tipo = rs.getInt("tipo");
        listaP.add(p);
        //Añadimos nuevo producto p
    }
    //PASO 6 - Se abre el puerto de conexion
    ServerSocket socket = new ServerSocket (12345);
    //PASO 7 - Esperando cliente
    Socket cliente = socket.accept();
    //PASO 8 - Enviamos los productos
    ObjectOutputStream bufferObjetos = new ObjectOutputStream (cliente.getOutputStream());
    bufferObjetos.writeObject(listaP);
    //PASO 9 - Cerramos conexiones
    bufferObjetos.close();
    cliente.close();
    socket.close();
```

Figura 3.6 – Servidor en Java

Ahora veremos (*Figura 3.7*) la parte del cliente como abre la conexión y recibe los productos para luego tratarnos como deseemos.

Cliente recibiendo productos:

```
public void SocketClienteRecibirProductos()
{
    try
    {
        Context ctx = getApplicationContext();
        //PASO 1 - Creamos el socket
        Socket socket = new Socket ("37.14.87.77", 12345);
        //PASO 2 - Obtenemos el grupo de productos
        ObjectInputStream bufferObjetos = new ObjectInputStream (socket.getInputStream());
        //PASO 3 - Tratamos cada producto recibido por separado
        ArrayList<Producto> listaP = new ArrayList<Producto>();
        listaP = (ArrayList<Producto>) bufferObjetos.readObject();
        int tam = listaP.size();
        for(int i = 0; i< tam; ++i){
            String msg = "Hemos recibido el producto: ";
            int duration= Toast.LENGTH_SHORT;
            Toast toast= Toast.makeText(ctx, msg + listaP.get(i).toString(), duration);
            toast.show();
        }
    }
    catch (Exception e)
    {
        MediaPlayer mp = MediaPlayer.create(this, R.raw.button5);
        mp.start();
        Context ctx = getApplicationContext();
        Toast toast= Toast.makeText(ctx, "No se ha podido conectar con el servidor.", Toast.LENGTH_SHORT);
        toast.show();
        e.printStackTrace();
    }
}
```

Figura 3.7 – Cliente en Android

A parte de los comentarios de ambos códigos paso a paso, lo único que cabe resaltar es que para enviar objetos entre sockets utilizaremos la clase *ObjectOutputStream* y para recibirlos usaremos *ObjectInputStream*. Es muy importante que el objeto que enviemos este definido en las dos partes de la comunicación y de la misma manera.

Para finalizar el tema de conexión vía internet es muy importante desde el servidor abrir los puertos del router por los que se van a crear los sockets. Por parte del cliente es imprescindible establecer en el documento AndroidManifest.xml los permisos de conexión a internet escribiendo:

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

3.4. Geolocalización

El programa también se encarga de localizar tu ubicación actual dentro del mapa de Google y además es capaz de trazar rutas desde esa ubicación hasta una ubicación destino.

Antes de empezar a utilizar el servicio de mapas de Google es necesario realizar algunas tareas previas. En primer lugar, debemos asegurarnos de que tenemos instalado las APIs de Google. Estos paquetes se llaman normalmente "Google APIs by Google, Android API x, revisión y". Además el documento AndroidManifest.xml debe contener los permisos de:

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

Para aceptar la conexión a internet.

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

Para permitir localización por redes.

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

Permitir localización por GPS.

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

Metemos la librería de Google Maps.

Para poder utilizar la API de Google Maps se requiere la obtención previa de una clave de uso (API Key), que estará asociada al certificado con el que firmamos digitalmente nuestra aplicación. Esto quiere decir que si cambiamos el certificado con el que firmamos nuestra aplicación tendremos que cambiar también la clave de uso de la API.

Una vez contamos con la clave de uso de la API, la inclusión de mapas en

nuestra aplicación es una tarea relativamente sencilla y directa. Para incluir un mapa de Google Maps en una XML de nuestra aplicación utilizaremos el *controMapView*. Tan solo hay que tener en cuenta que tendremos que indicar nuestra clave de uso de Google Maps en su propiedad *android:apiKey* y ya podremos verlo en pantalla. Su código seria así (*Figura 3.8*):

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

<com.google.android.maps.MapView
    android:id="@+id/mapa"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:apiKey="0ss-5q6s3FKYkkp3atMUH..."
    android:clickable="true" />

</LinearLayout>
```

Figura 3.8 – Lienzo para mapas

Una vez creado el mapa, podremos navegar con él y mostrar nuestra ubicación. Pero si lo que deseamos es trazar rutas, calcular distancias, etc. deberemos recoger la longitud y latitud actual para efectuar dichos cálculos.

```
LocationManager locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
// Lista de proveedores
List<String> providers = locationManager.getAllProviders();

Criteria criteria = new Criteria();
String bestProvider = locationManager.getBestProvider(criteria, false);


Location location = locationManager.getLastKnownLocation(bestProvider);
if(location != null){
    double lat = location.getLatitude();
    double lon = location.getLongitude();
}
```

Figura 3.9 – Obteniendo ubicación actual

Una vez obtenida la longitud y latitud ya podemos hacer los cálculos de distancias o trazabilidad con las funciones que nos ofrece Google Maps.

3.5. Sonidos multimedia

Actualmente, las características de los teléfonos que vienen con Android nos permiten consumir contenidos multimedia sin que el dispositivo sufra al cumplir con esta tarea. Vamos a conocer las dos APIs que Android provee para trabajar con sonidos: *SoundPool* y *MediaPlayer*.

Podemos utilizar *SoundPool* para reproducir pequeñas pistas de audio. Con esta clase podemos repetir reproducción de sonidos y hasta reproducir múltiples sonidos de manera simultanea. Un dato importante a tener en cuenta cuando se trabaja con esta clase, es que los archivos de sonido que quieras reproducir no deben sobrepasar 1MB de tamaño.

Básicamente, esta clase carga el archivo de forma asíncrona; además de que a partir de la versión Android 2.2 es posible comprobar si la carga se ha completado a través de un objeto *OnLoadCompleteListener*.

También resulta interesante saber, que el botón de volumen del teléfono puede ser configurado para controlar un *stream* de audio en específico. Para conseguir esto es necesario especificar el tipo de audio que maneja nuestra aplicación. Las siguientes líneas (*Figura 3.10*) de código nos ayudarían a controlar el flujo de sonido multimedia que estemos escuchando en nuestra aplicación tanto si queremos subir o bajar el volumen.

```
audio = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
```

Figura 3.10 – Crear servicio de audio

Primero hemos declarado el servicio que utilizaremos y después lanzamos la función que recoge la configuración de sonido.

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    switch (keyCode) {
    case KeyEvent.KEYCODE_VOLUME_UP:
        audio.adjustStreamVolume(AudioManager.STREAM_MUSIC,
                AudioManager.ADJUST_RAISE, AudioManager.FLAG_SHOW_UI);
        return true;
    case KeyEvent.KEYCODE_VOLUME_DOWN:
        audio.adjustStreamVolume(AudioManager.STREAM_MUSIC,
                AudioManager.ADJUST_LOWER, AudioManager.FLAG_SHOW_UI);
        return true;
    default:
        return false;
    }
}
```

Figura 3.11 – Funciones de control de volumen

Sin embargo, como todo servicio extra en un programa, es necesario atribuir servicios extras en el documento AndroidManifest.xml mediante la línea (*Figura 3.12*):

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

Figura 3.12 – Aceptar permisos de audio

Cuando queramos trabajar con pistas de audio más largas o videos, la clase *MediaPlayer* será la elección a trabajar. Esta receta es bastante simple para reproducir clips de audio en Android. Basta con usar el método *create(context, int)* de la clase *MediaPlayer*, pasándole como parámetro la actividad desde donde se reproduce el sonido y el archivo de referencia de audio/video a reproducir. Tan sencillo como esto:

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.button1);
mp.start();
mp.stop();
mp.release();
```

Figura 3.13 – Funciones estándar de la clase MediaPlayer

En este caso, *R.raw.button1* representa el archivo audio con nombre "button1.mp3" almacenado en la carpeta "*res/raw*". Las función *start()* inicia el sonido, la función *stop()* lo detiene y la función *release()* libera el espacio en memoria que esté ocupando.

4. Especificación EasyMarket

En el siguiente capítulo vamos a comprender en que consiste completamente el programa definiendo todas sus partes. También veremos de que forma ha sido estructurado y explicaré de forma detallada las dos funciones que necesitan un mayor entendimiento.

4.1. Definición de la aplicación

EasyMarket es una aplicación de simular compras que lo que pretende conseguir es que el usuario tenga un conocimiento de los supermercados sin tener que ir a ellos directamente a consultarlos y además le proporciona la oportunidad de, si desea efectuar una compra, poder efectuar una opinión de qué centro sería el más idóneo para él estableciendo unas preferencias iniciales en el programa.

Después de esta definición breve, vamos a ver más profundamente todas las características que tiene la aplicación paso a paso y mostrándolo visualmente para hacer más fácil su comprensión.

4.1.1. Iniciando la aplicación

La aplicación al ser ejecutada desde el icono que mostramos en la *Figura 4.1* nos lleva a una pantalla de carga (*Figura 4.2*) la cual podemos ver como empieza su ejecución. He escogido este titulo porque he creído que era bastante intuitivo a la hora de explicar con una sola palabra el concepto del programa.

Figura 4.1 – Icono aplicación

Figura 4.2 – Pantalla de carga

4.1.2. Menú principal

Tras varios segundos de carga el programa nos llevará directamente al menú general. Desde aquí podremos tanto navegar entre todas las opciones como salir de la aplicación. También podremos actualizar la base de datos del sistema o ver la lista de productos y centros de una manera más rápida.

Figura 4.3 – Pantalla Menú

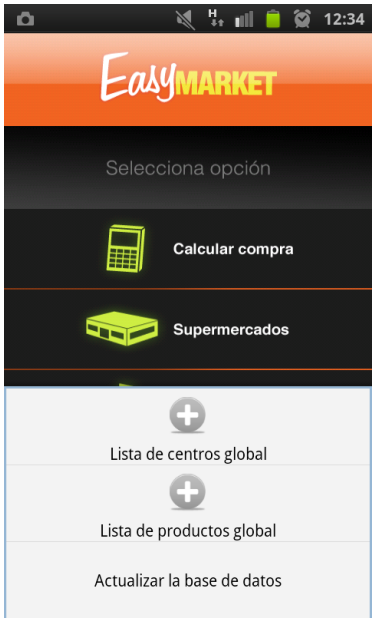


Figura 4.4 – Pantalla Menú 2

4.1.2.1. Supermercados

En la siguiente pantalla (*Figura 4.5*) podemos observar las distintas características que nos ofrece el programa. Si deseamos mostrar o eliminar un programa deberemos seleccionarlo previamente en la lista que nos ofrece esta pantalla.

Figura 4.5 – Pantalla Menú Centros

4.1.2.1.1. Mostrar centro

Una vez seleccionado el supermercado a mostrar podremos editar todos sus aspectos dentro de esta pantalla. El programa nos ofrece tanto cambiar su nombre como las valoraciones. Además nos muestra los productos que el centro posee. También podemos hacer comentarios o ver las valoraciones que otras personas desde otros dispositivos han hecho sobre este centro.

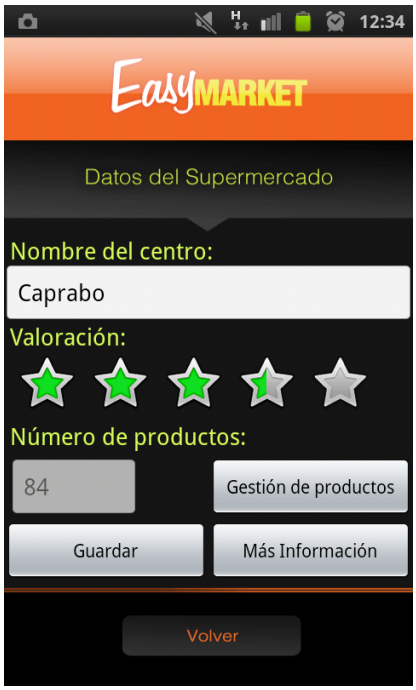


Figura 4.6 – Información Centro 1

Figura 4.7 – Información Centro 2

El centro posee una lista de ubicaciones registradas en las que se encuentra (*Figura 4.8*). Lo he restringido a toda Cataluña inicialmente. Con el mapa podremos ver donde esta el centro seleccionado y como podríamos llegar hasta él desde nuestra ubicación actual.

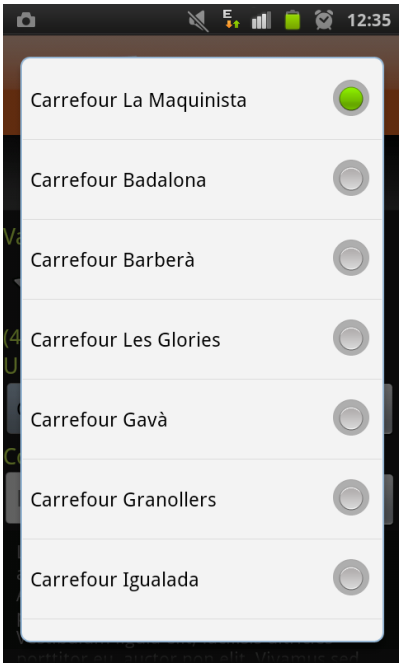


Figura 4.8 – Lista Ubicaciones

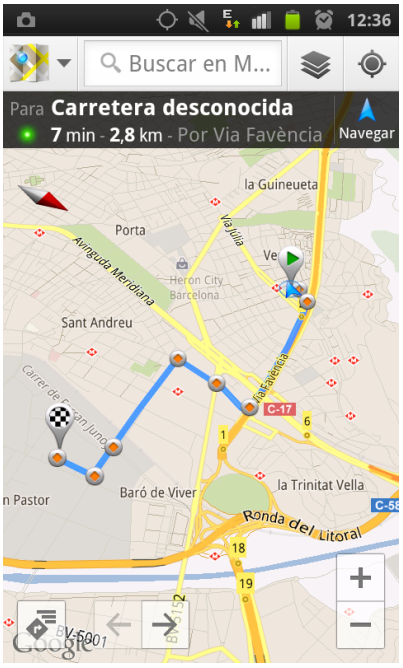


Figura 4.9 – Ruta de centro

Finalmente y no por ello menos importante, dentro de este menú de edición podremos añadir o quitar productos del centro (*Figura 4.11*). Los que se encuentren en verde serán aquellos que el centro posee y los que están en rojo serán los que existen en el sistema de la aplicación pero el centro no los tiene registrados (*Figura 4.10*). Si por alguna razón buscamos introducir un producto que no se encuentra en el sistema desde este menú lo podremos hacer directamente. Para facilitar la usabilidad del programa también podremos "Añadir al carrito" el producto deseado si quisiéramos hacerlo desde aquí.

Figura 4.10 – Lista de productos del centro

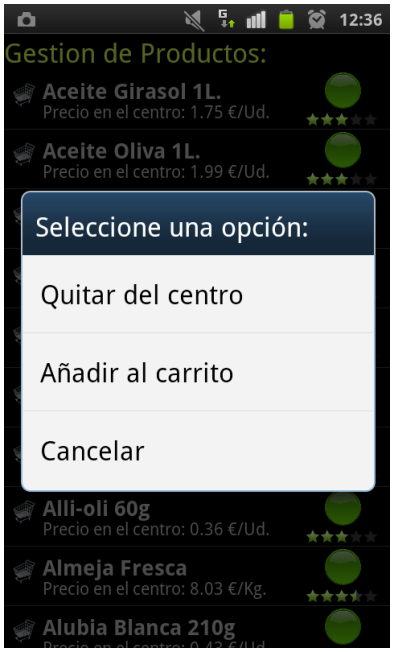


Figura 4.11 – Añadir producto al centro

4.1.2.1.2. Eliminar centro

El centro se eliminará del sistema borrando las ubicaciones que tenía guardadas. Es muy importante estar seguro de esta acción puesto que no es reversible a nos ser que reinstalemos el programa.

4.1.2.1.3. Crear centro

Para crear un centro simplemente introduciremos su nombre y valoración. A continuación nos preguntará si deseamos añadir productos directamente. El programa compartirá toda la información relativa a este centro cuando se actualice si el servidor guarda alguna información con un centro del mismo nombre.

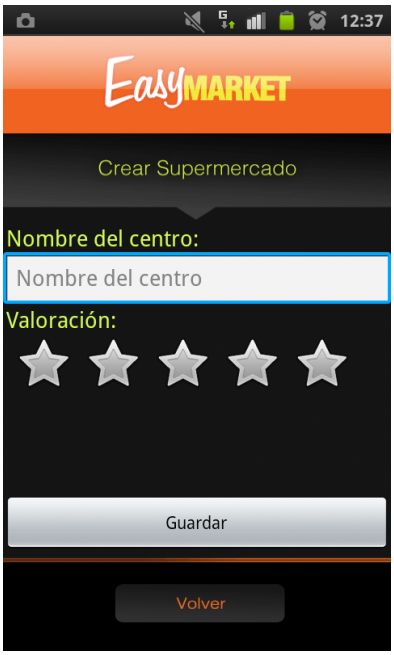


Figura 4.12 – Pantalla crear centro

4.1.2.2. Productos

En la siguiente pantalla (*Figura 4.13*) podemos observar las distintas características que nos ofrece el programa. Si deseamos mostrar o eliminar un producto deberemos seleccionarlo previamente en la lista que nos ofrece esta pantalla.

Figura 4.13 – Pantalla Menú Productos

4.1.2.2.1. Mostrar Producto

Una vez seleccionado el producto a mostrar podremos editar todos sus aspectos dentro de esta pantalla. El programa nos ofrece tanto cambiar su nombre como la unidad de medida en la que ha sido guardado. Además nos muestra la puntuación global que recibe de todos los centros en los que está. También podemos hacer comentarios o ver las valoraciones que otras personas desde otros dispositivos han hecho sobre este producto.

Figura 4.14 – Pantalla datos producto

Dentro de este menú de edición podremos añadir o quitarlo de los centros que deseemos. Los centros que se encuentran en verde son aquellos que tienen dicho producto y los que están en rojo por lo contrario son aquellos que no (*Figura 4.15*). Si vemos que no existe el centro al que queremos añadirlo desde esta pantalla podremos ir directamente a la creación del centro.

Figura 4.15 – Lista de centros del producto

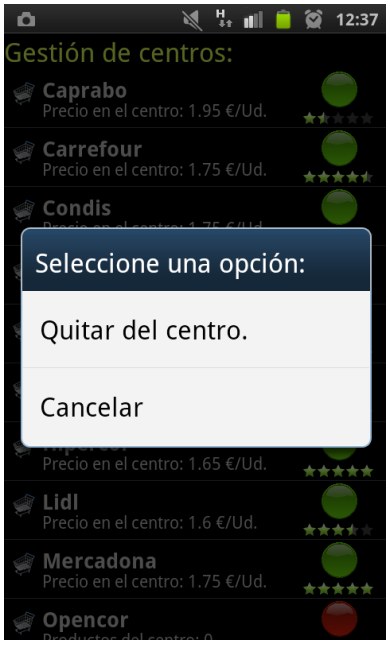


Figura 4.16 – Añadir producto al centro

Para mejorar la usabilidad también he añadido una opción rápida desde este menú para que el usuario pueda añadir el producto directamente al carrito.

4.1.2.2.2. Eliminar Producto

El producto se eliminará del sistema eliminando todos los comentarios. Es muy importante estar seguro de esta acción puesto que no es reversible a nos ser que reinstalemos el programa.

4.1.2.2.3. Crear Producto

Para crear un producto deberemos introducir su nombre y su unidad de medida. A continuación, si así lo queremos, podremos añadirlo a los centros en los que se encuentra.

Figura 4.17 – Pantalla crear producto

4.1.2.3. Calcular Compra

Ahora veremos como el usuario puede hacer la compra para obtener los centros que sean más afines con sus preferencias. Para ello explicaré los 3 pasos que hay.

4.1.2.3.1. Definir productos del carrito

En esta pantalla (*Figura 4.18*) el comprador podrá añadir los productos o cargar listas de compra que ya haya guardado en ocasiones anteriores. Si ha añadido productos previamente el sistema los tendrá guardados ya en la compra actual.

Figura 4.18 – Pantalla añadir productos al carrito

4.1.2.3.2. Establecer preferencias de compra

Una vez escogidos los productos y sus cantidades respectivas, la persona debe establecer las preferencias que más importancia tengan para él según 3 criterios: el precio de la compra, la proximidad del centro o la calidad del centro.

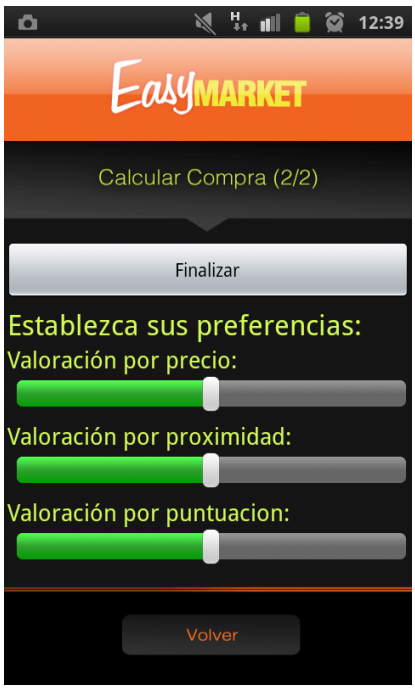


Figura 4.19 – Establecer preferencias

4.1.2.3.3. Centros preferidos

Cuando el cliente tenga establecidas sus gustos, el siguiente paso le mostrara los 3 mejores centros que el sistema le recomienda comprar. En este menú podremos ver el precio de la comprar total que le saldría, la puntuación que tiene cada centro y donde estaría su ubicación más cercana respecto a él.

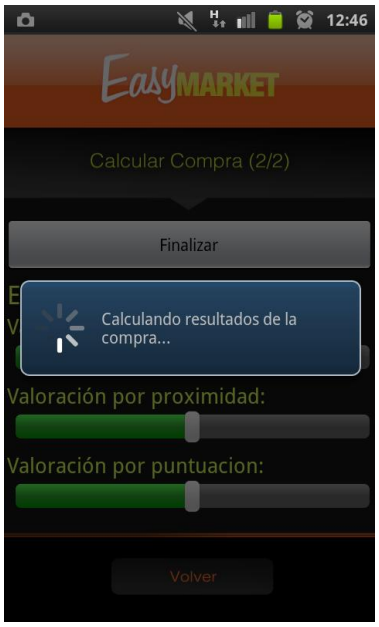


Figura 4.20 – Calculando compra

Figura 4.21 – Centros recomendados 1

Figura 4.22 – Centros recomendados 2

Tenemos la posibilidad de guardar esta lista de los productos seleccionados por si es una compra habitual y deseamos consultarla en ocasiones futuras.

4.1.2.4. Localización

El programa también ofrece la facilidad de mostrarnos un mapa para saber nuestra ubicación y por si queremos consultar de manera libre algún lugar del mapa que nos interese.

Figura 4.23 – Pantalla ubicación

4.2. Diagramas de casos de uso

En las siguientes imágenes podremos ver las acciones principales que el usuario podrá realizar dentro de la aplicación. Las he dividido en los tres bloques más importantes que son los tres tipos de servicio que podrá ofrecer: gestión de productos, gestión de centros y gestión de compra.

A continuación veremos con más detalle la descripción de los casos de uso nombrados previamente.

4.2.1. Gestión de productos

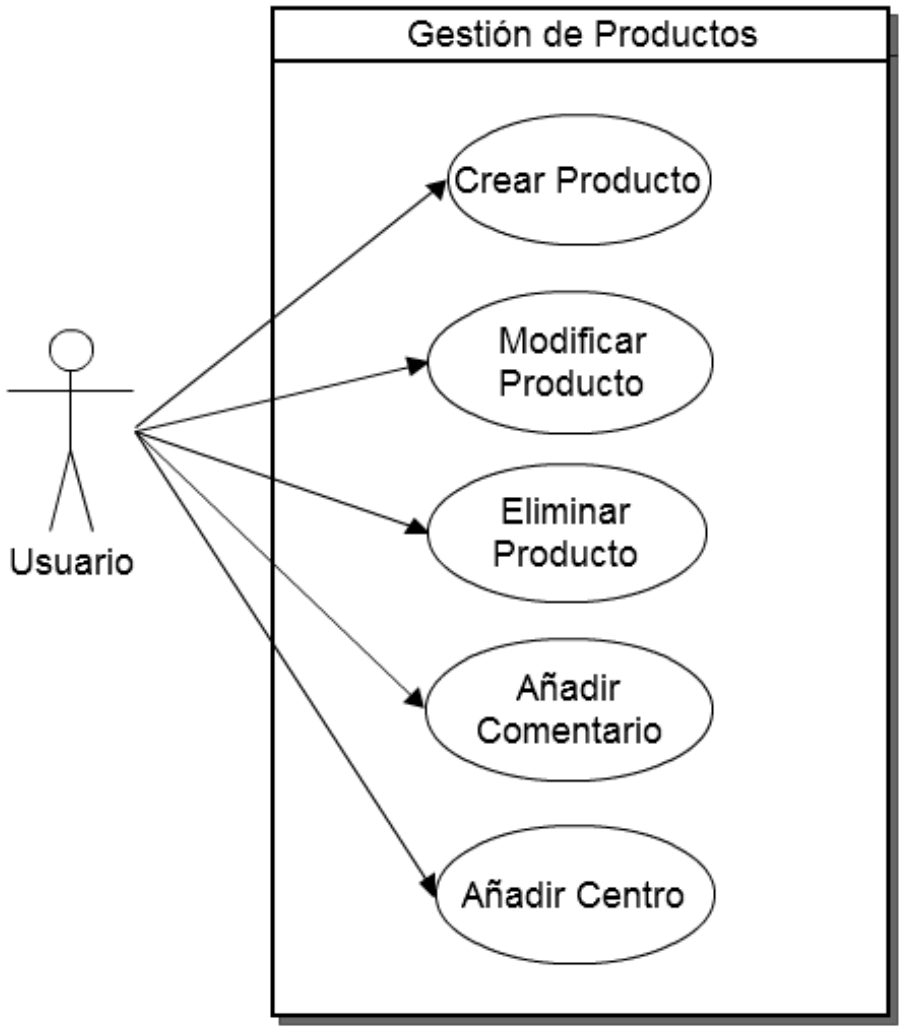


Figura 4.24 – Caso de uso Gestión de productos

Desde el “Menú de Productos” el propietario del dispositivo podrá hacer ciertas acciones que describiré a continuación:

- **Caso de uso “Crear Producto”**

Actores: Usuarios
Descripción: Para crear un producto el usuario deberá introducir el nombre de este y la unidad de medida en la que será contada.

- **Caso de uso “Modificar Producto”**

Actores: Usuarios
Descripción: El producto puede ser modificado ya sea para cambiar su nombre, unidad de medida o realizar alguna gestión con los centros.

- **Caso de uso “Eliminar Mercado”**

Actores: Usuarios
Descripción: El usuario podrá eliminar el producto creado.

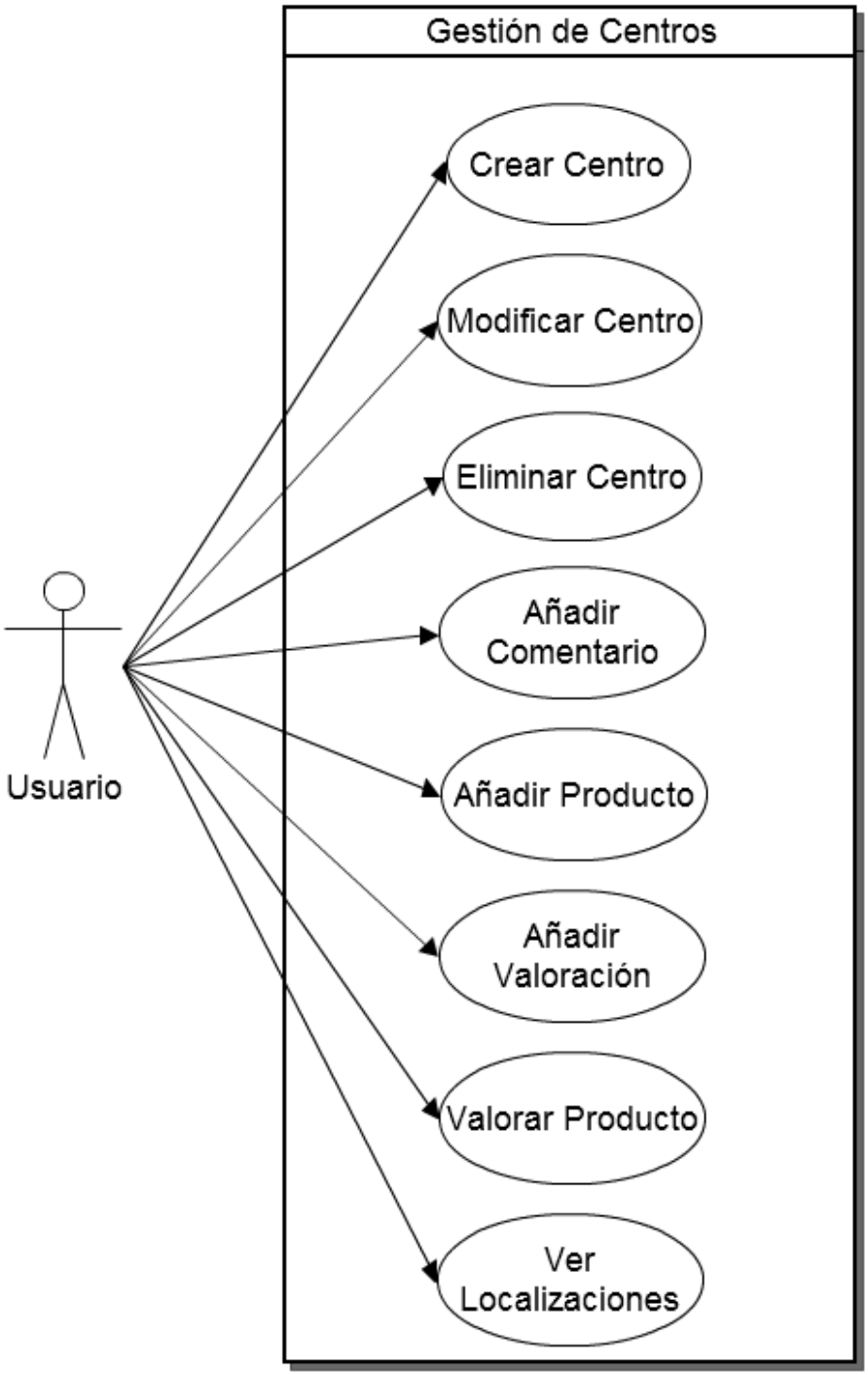
- **Caso de uso “Comentar Producto”**

Actores: Usuarios
Descripción: También es posible comentar cualquier producto con tal de compartir las ideas para tenerlas en cuenta otros usuarios de la aplicación.

- **Caso de uso “Añadir/Quitar Producto a Centros”**

Actores: Usuarios
Descripción: La persona debe seleccionar el centro y poner una puntuación y un precio para tal centro. Por otro lado también podrá quitarlo de los centros que crea conveniente perdiendo el precio y la valoración que tenían asignados.

4.2.2. Gestión de centros



- Figura 4.25 – Caso de uso Gestión de centros

- **Caso de uso "Crear Centro"**

Actores: Usuarios
Descripción: Para crear un centro la persona deberá introducir el nombre y la puntuación de calidad que crea conveniente que debe tener.

- **Caso de uso "Modificar Centro"**

Actores: Usuarios
Descripción: Cualquier centro puede ser modificado ya sea para cambiar su nombre, su valoración personal o realizar cualquier gestión con los productos del sistema.

- **Caso de uso "Eliminar Centro"**

Actores: Usuarios
Descripción: El usuario podrá cualquier centro de la aplicación teniendo en cuenta que si este contiene geolozalizaciones estas serán perdidas así como el resto de los productos con sus respectivos precios asignados.

- **Caso de uso "Comentar Centro"**

Actores: Usuarios
Descripción: Se podrá hacer una crítica al centro para luego ser distribuida en todos los dispositivos que contengan la aplicación.

- **Caso de uso "Valorar Centro"**

Actores: Usuarios
Descripción: El usuario podrá valorar cada centro y de esa manera compartirlo con otras personas para tener un criterio más amplio y exacto de este.

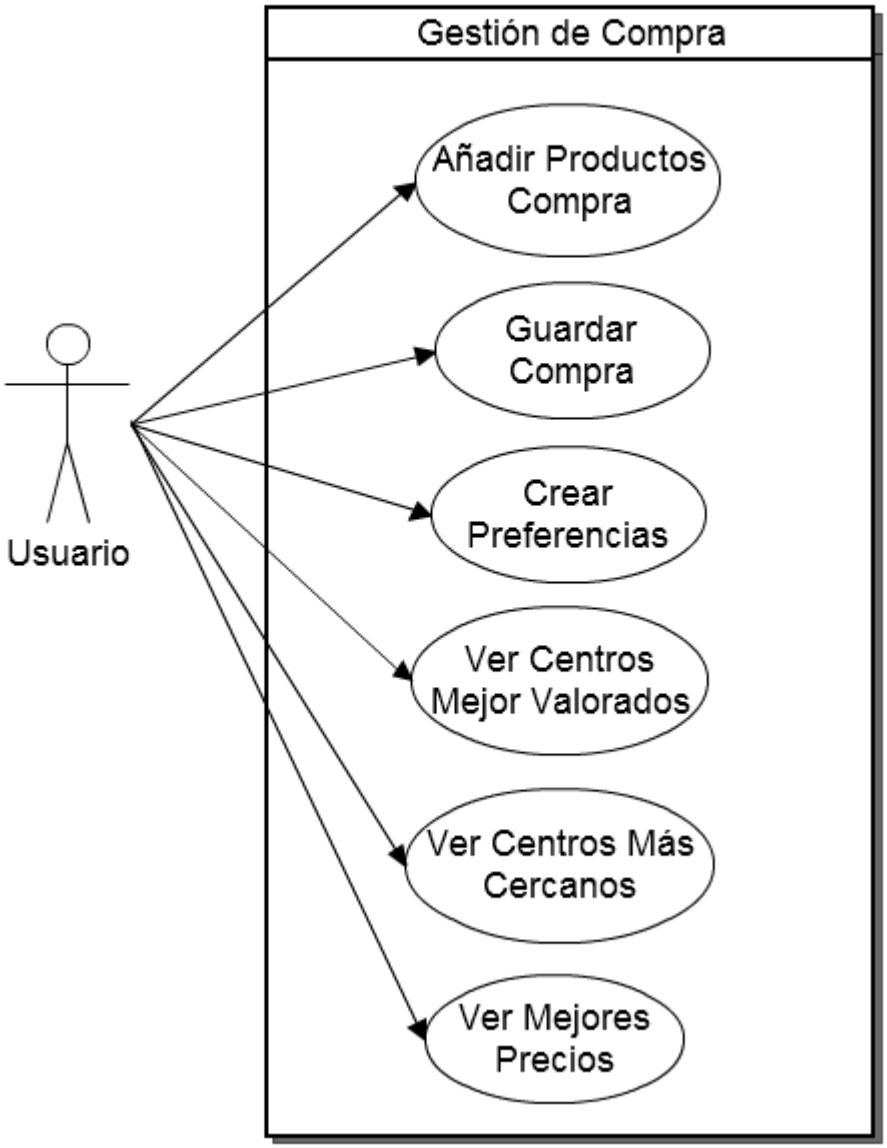
- **Caso de uso "Añadir/Quitar Producto a Centro"**

Actores: Usuarios
Descripción: También se podrán añadir los productos con un precio y una valoración de calidad según el criterio personal. De la misma manera se podrán quitar si este centro ya no contiene dichos productos.

- **Caso de uso "Ver Localización"**

Actores: Usuarios
Descripción: El centro puede tener registradas múltiples direcciones a las que el usuario podrá acceder si lo desea para que le sea trazada una ruta desde el punto en el que se encuentra actualmente.

4.2.3. Gestión de compra



- Figura 4.26 – Caso de uso Gestión de compra

- **Caso de uso "Añadir Producto a la Compra"**

Actores: Usuarios
Descripción: Este es el primer paso para realizar la compra. La persona debe seleccionar los productos que quiere agregar al "carrito" así como sus cantidades respectivas.

- **Caso de uso "Establecer Preferencias"**

Actores: Usuarios
Descripción: El usuario seleccionará la importancia que quiere darle a los criterios de "Precio", "Puntación del centro" y "Cercanía del centro" según crea más conveniente para él.

- **Caso de uso "Mejores Estadísticas"**

Actores: Usuario
Descripción: Una vez realizado los dos pasos anteriores el comprador podrá ver que centros son los que más se ajustan a sus gustos. Podrá consultar donde están situados en el mapa si lo desea así como ver sus precios y puntuaciones.

- **Caso de uso "Guardar Lista Compra"**

Actores: Usuario
Descripción: Al final de la gestión de compra se puede guardar la lista de los productos con sus cantidades respectivas por si el responsable pretende usarla en compras futuras.

4.3. UMLs

4.3.1. Activities

En el siguiente UMLs podremos ver las pantallas y los activities que he creado. He intentado optimizar de la mejor manera que he podido las pantallas para evitar código duplicado. La eficiencia en las aplicaciones móviles es una característica muy importante.

En este modelo podremos ver también las navegabilidades que tienen entre ellos con las funciones que contienen. El modelo intenta describir las características más necesarias para poder navegar entre ellos y ofrecer os principales servicios.

Como ya hemos explicado anteriormente, los XML representan las pantallas mostradas en el dispositivo. Los Activities definen las interacciones que efectuará cada pantalla.

El programa empieza con el *Activity* "*loading*" el cual se encarga de abrir la base de datos y configurar los parámetros de la aplicación. A continuación nos lleva a "*MainSelection*" la cual es la pantalla principal y a partir de ahí el usuario puede navegar hacia múltiples sitios según lo que desee.

UML Capa Presentación (Activities y Navegabilidades)

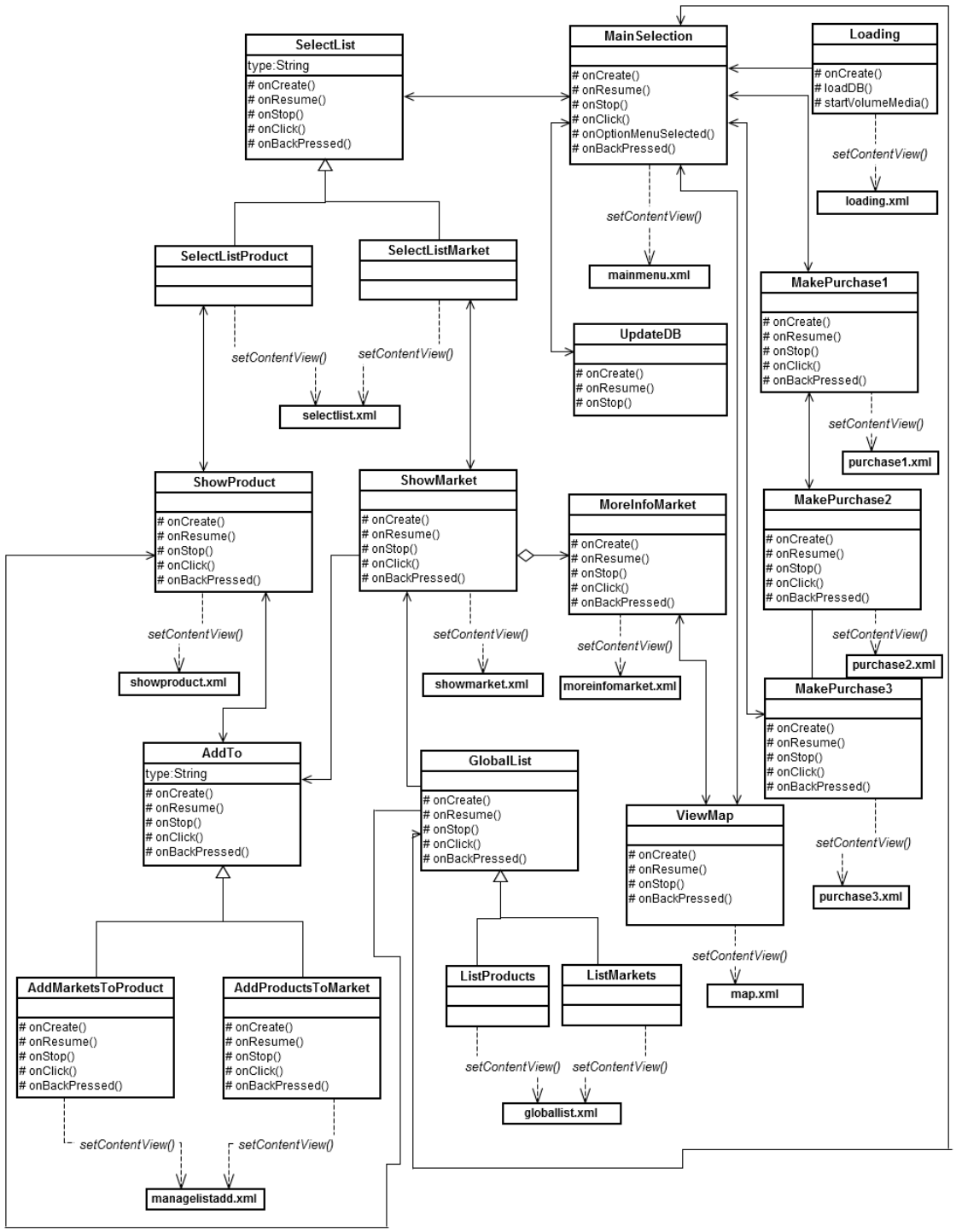


Figura 4.27 – Modelo Activities y XMLs

4.3.2. Orientación a objetos

Puesto que la aplicación se basa gestionar productos y centros he creído muy conveniente describir los objetos que la forman ya que así podremos entender qué tenemos registrado. Es una aplicación en la cual es muy conveniente utilizar *Transaction Script* debido a que el programa está constantemente usando consultas a la Base de Datos. Para evitar perdidas de información por cierres indebidos que ocurren frecuentemente en los dispositivos móviles la mejor manera de guardar todos los datos es en el SGDB.

La capa de dominio en este caso es bastante simple ya que simplemente se encarga de efectuar las consultas para mostrar por pantalla la información deseada. Únicamente esta capa se vuelve más compleja en el momento de efectuar la compra (Caso de uso de “Gestión de Compra”). En este momento es cuando debe efectuar cálculos de ponderación y trazabilidad para mostrar la información.

Cabe decir que, como podremos observar, todos los objetos contiene un “_id” el cual referencia a un objeto de esa clase. En Android todos los objetos creados en la Base de Datos contienen un _id obligatorio el cual se debe declarar. Como es lógico he reutilizado este id para declararlo en muchas ocasiones como clave única de la clase.

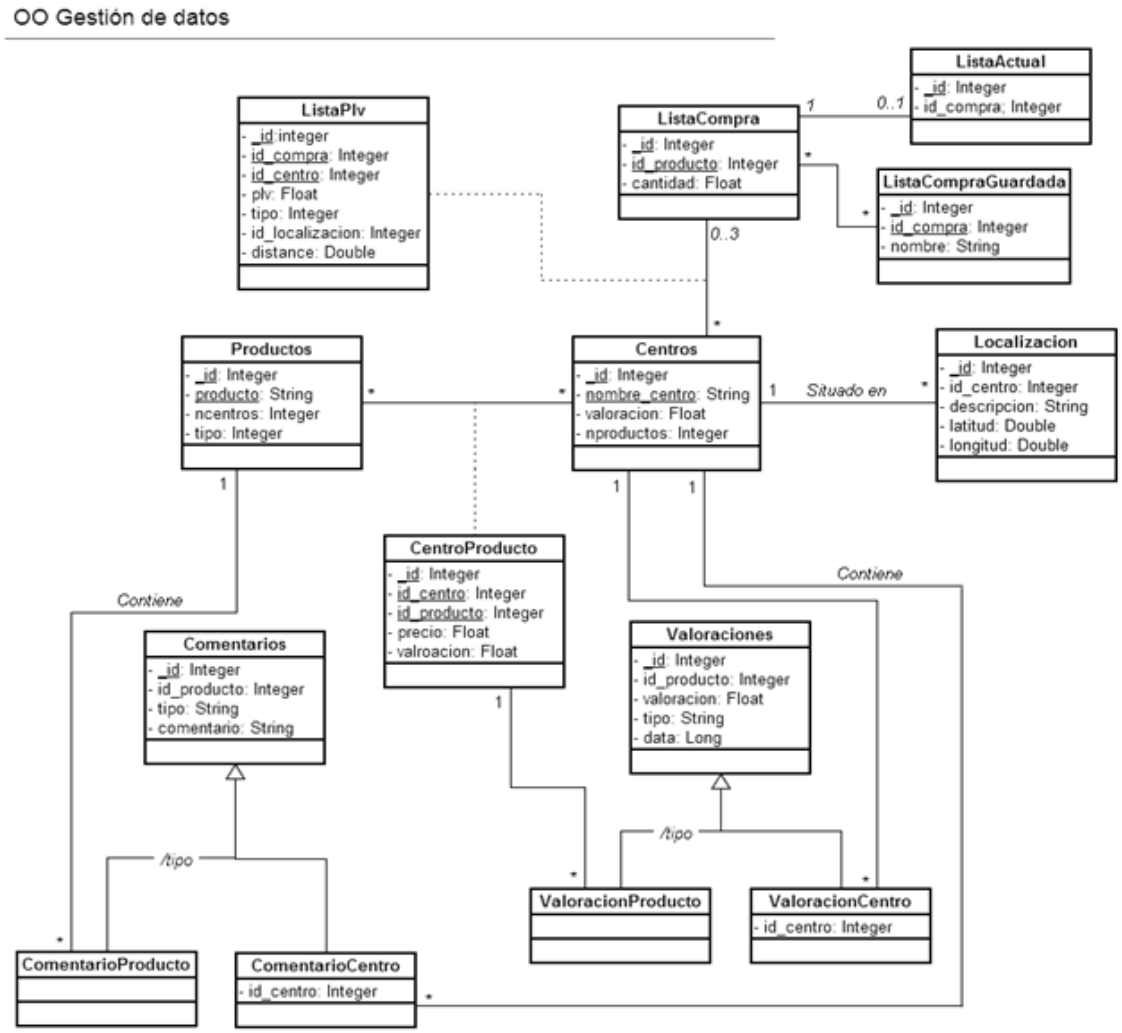


Figura 4.28 – Modelo de objetos

4.4. Funciones principales

En este apartado he descrito dos funciones internas del programa que creo interesante comprender para entender un poco mejor como funciona la aplicación.

4.4.1. Ponderar calificaciones

Los centros, así como los productos, tienen valoraciones que la gente comparte según la valoración que creen ellos de manera subjetiva. Sin embargo un centro puede cambiar a lo largo del tiempo y no siempre mantiene su estatus de calidad tanto para bien como para mal. Por esa razón cada valoración que una persona comparte queda registrada con una data.

Esta función lo que pretende es que las valoraciones de los centros tengan diferentes ponderaciones en función del momento en que se hayan efectuado. Para ellos he pensado que los centros cambian su estilo o su ética de empresa cada año. Por tanto las ponderaciones que he creado son:

- **Puntuaciones con menos de 1 año de antigüedad:** Estas puntuaciones tendrán un 100% de su valor en función de todas las demás ya que son las más recientes y el usuario es muy probable que esté en lo cierto con la calidad actual de la empresa.
- **Puntuaciones realizadas entre 1 y 2 años de antigüedad:** Las valoraciones puede que hayan perdido cierto criterio ya que es muy probable que la empresa haya cambiado su estructura y ya no mantenga la misma calidad que en aquellos momentos. Por esa razón le he atribuido un 66% de valor en función del global de todas las calificaciones.
- **Puntuaciones realizadas entre 2 y 3 años de antigüedad:** Estas puntuaciones recibirán un decremento mayor de su importancia puesto que tienen más tiempo y es aun más probable que el supermercado no mantenga el mismo estatus. Tendrán una importancia del 33% en función del global.
- **Puntuaciones realizadas con más de 3 años de antigüedad:** Las calificaciones con dicha antigüedad tendrán el más mínimo valor ya que es improbable que coincida el criterio que efectuaron los usuarios en ese momento con la situación actual del centro. Las puntuaciones tendrán un valor del 10%.

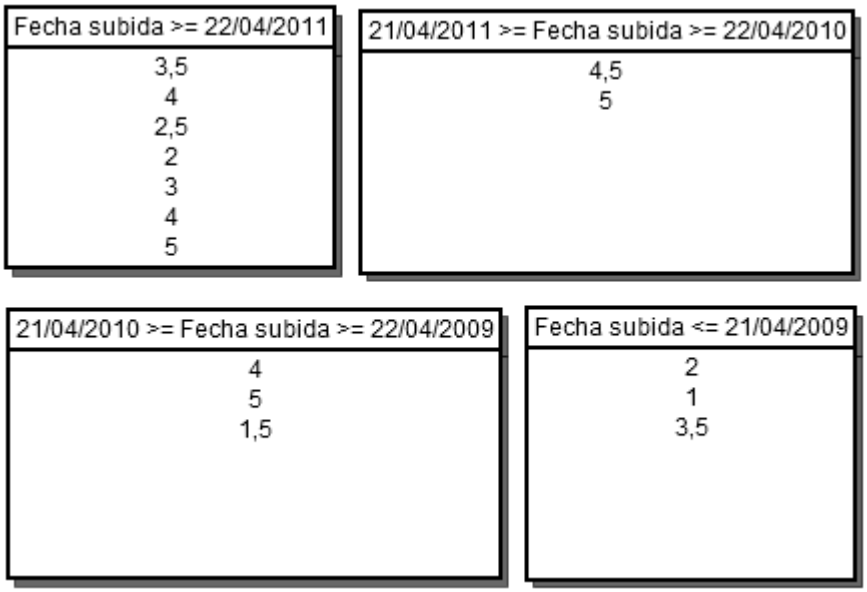
Para comprender mejor este método pondremos un ejemplo simple de valoraciones con un supermercado y veremos de qué manera he construido esta función. Las datas de valoraciones las he guardados en milisegundos para ser más exacto por lo que un año equivale a 31.536.000.000 milisegundos aproximadamente. Sin embargo, para entender mejor el ejemplo mostraré las datas en formato *dd/mm/aaaa* para que no haya confusión de números. Las valoraciones son del 0 al 5 teniendo en cuenta que el 0 es la peor puntuación y 5 la mejor.

Ejemplo*: Imaginemos que tenemos un centro llamado "Supermarket X" al cual se le han efectuado 7 valoraciones. Estamos a día 22 de abril del 2012 y queremos mostrar la valoración global del centro. Las correspondientes valoraciones son:*

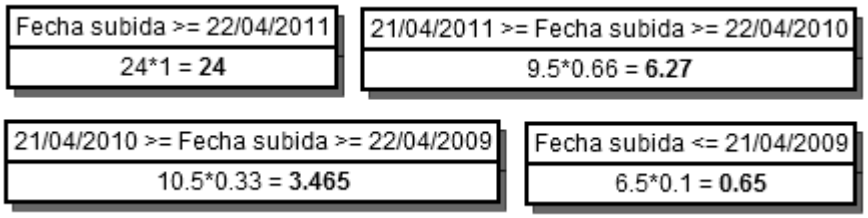
- **Puntuación**: 3.5, **Fecha de valoración**: 15/04/2012
- **Puntuación**: 4, **Fecha de valoración**: 22/01/2012
- **Puntuación**: 4, **Fecha de valoración**: 11/12/2009
- **Puntuación**: 4.5, **Fecha de valoración**: 15/02/2012
- **Puntuación**: 5, **Fecha de valoración**: 22/01/2011
- **Puntuación**: 5, **Fecha de valoración**: 28/02/2010
- **Puntuación**: 2.5, **Fecha de valoración**: 20/04/2012
- **Puntuación**: 2, **Fecha de valoración**: 02/08/2008
- **Puntuación**: 1.5, **Fecha de valoración**: 22/01/2010
- **Puntuación**: 2, **Fecha de valoración**: 10/10/2011
- **Puntuación**: 1, **Fecha de valoración**: 29/03/2007
- **Puntuación**: 3, **Fecha de valoración**: 22/01/2012
- **Puntuación**: 4, **Fecha de valoración**: 09/03/2012
- **Puntuación**: 3.5, **Fecha de valoración**: 02/01/2009
- **Puntuación**: 5, **Fecha de valoración**: 14/11/2011

Para empezar las agruparemos en los 4 grupos según la fecha en la que los usuarios hicieron cada valoración.

Data actual: 22/04/2012



Una vez agrupados haremos el sumatorio en cada grupo y lo multiplicaremos por el porcentaje de importancia según el tiempo.



Con estos valores haremos una suma y a continuación los dividiremos por el nº de votos que tiene cada uno multiplicados por su porcentaje correspondiente.

$$\frac{24 + 6.27 + 3.465 + 0.65}{(7*1) + (2*0.66) + (3*0.33) + (3*0.1)} = \frac{34.385}{9.61} = 3.578$$

*Como resultado la puntuación global para Supermarket X es **3.578**.*

La formula que he aplicado seria:

$$\frac{\Sigma(\text{1er Grupo} * 1) + \Sigma(\text{2o Grupo} * 0.66) + \Sigma(\text{3er Grupo} * 0.33) + \Sigma(\text{4o Grupo} * 0.1)}{(\text{Nº votos Grupo 1} * 1) + (\text{Nº votos Grupo 2} * 0.66) + (\text{Nº votos Grupo 3} * 0.33) + (\text{Nº votos Grupo 4} * 0.1)}$$

4.4.2. Búsqueda de compra más adecuada

La siguiente función lo que pretende es buscar los centros que más se adecúan al usuario una vez él ha seleccionado los productos y ha elegido las preferencias que mas le convenían.

El programa busca que centros contienen los productos que ha seleccionado y calcula el precio que le costaría en dicho lugar. A continuación calcula la media de puntuaciones de ese centro por medio de la función que hemos explicado en el apartado anterior y lo siguiente es mirar la distancia más próxima de cada supermercado en referencia a la posición actual del comprador.

Una vez obtenidos el precio, la valoración y la ubicación más próxima de cada centro que contiene los productos añadidos, lo que se hace es coger ponderar cada uno de los 3 atributos en función de: el precio más caro, la ubicación más lejana y la puntuación más alta.

Las preferencias que el usuario ha establecido de precio, valoración y proximidad se barajan entre el 0 y el 1. Estos valores serán multiplicados por cada atributo del centro y finalmente se hará una suma de todos. Los 3 centros que obtengan la mejor puntuación serán las recomendadas al cliente.

Cogiendo esos 3 tipos de valores calcularemos sus valores de la siguiente manera. Cabe decir que para cada bloque (precio, valoración, ubicación) la puntuación máxima que puede obtener es de 100. Por tanto en el mejor de los casos un centro podrá obtener como máximo 300 puntos. Cada bloque lo calculamos de la siguiente manera:

- **Precio:** Para obtener la puntuación que recibe cada centro en función de su precio partiremos atribuyéndoles a todos 100 puntos iniciales. A partir de ahí cogeremos el centro más caro y le atribuiremos 100 puntos por valor “negativo” al tener el precio más elevado y se lo restaremos a la puntuación inicial que tiene. En tal caso el centro con el precio más caro tendrá 0 puntos en el apartado de precio. Basándonos en la compra más cara

haremos una regla de tres con las siguientes compras y con ese valor obtenido se los restaremos a los 100 puntos iniciales que tiene cada uno. Cada puntuación final la multiplicaremos por la importancia que el usuario ha dado a este atributo (como ya hemos dicho estos atributos se valoran entre el 0 y 1).

- **Ubicación:** El proceso para evaluar las ubicaciones se calcula de la misma manera que el caso de los precios. Inicialmente le atribuimos 100 puntos a todas las localizaciones más cercanas de cada centro y partiendo de la ubicación más lejana (a la cual le damos 100 ptos "negativos" de valor) vamos calculando por reglas de tres todas las demás. Ese valor se lo restaremos a los 100 puntos iniciales y observaremos que la ubicación del centro mas lejano tendrá 0 puntos. Más tarde le multiplicaremos la ponderación que el usuario ha creído conveniente.
- **Puntaciones:** El caso de las puntuaciones es algo distinto y a la vez más sencillo. Sabiendo que las puntuaciones están limitadas entre 0 y 5, asignaremos 100 puntos a la valoración de 5 estrellas. Partiendo de ahí y con una regla de 3 asignaremos la puntuación que corresponda a cada valoración.

Como he resumido previamente, una vez calculado estas 3 maneras se sumaran en cada centro y los 3 que tengan mejor puntuación global serán los que recomendaremos en el sistema. Para verlo mejor explicaremos un caso sencillo con un ejemplo.

Ejemplo: *Un usuario introduce los productos al carrito con sus correspondientes cantidades. A continuación establece sus preferencias según la importancia que él le da al precio de la compra, a la cercanía de los centros y a la calidad que posee. Imaginemos que han sido 4 centros los que contienen todos los productos que el cliente desea. Una vez calculado el precio de la compra de cada centro, la ubicación más cercana de cada uno y su valoración de calidad, los datos obtenidos son los siguientes:*

Preferencias del usuario:
- **Precio: 0.7**
- **Proximidad: 0.5**
- **Calidad: 0.7**

Super X:
- **Precio compra:** 10.75€
- **Distancia:** 725 m
- **Valoracion:** 3.5

Super Y:
- **Precio compra:** 8.25€
- **Distancia:** 1025 m
- **Valoracion:** 3

Super W:
- **Precio compra:** 11€
- **Distancia:** 585 m
- **Valoracion:** 5

Super Z:
- **Precio compra:** 10.5€
- **Distancia:** 300 m
- **Valoracion:** 1

Ahora efectuaremos las puntuaciones según cada bloque y lo multiplicaremos por el indicador de importancia del usuario. Cabe recordar que tanto el bloque de precio como el bloque de distancia asignaremos 100 puntos iniciales ya que partiremos desde el peor caso y restaremos su valor equivalente:

Precio:
Super. W -- 11€ => 100 - 100 = 0 ptos => 0*0.7 = 0 ptos
Super. X -- 10.75€ => 100 - 97.72 = 2.28 ptos => 2.28*0.7 = 1.596 ptos
Super. Y -- 10.5€ => 100 - 95.45 = 4.45 ptos => 4.45*0.7 = 3.115 ptos
Super. Z -- 8.25€ => 100 - 75 = 25 ptos => 25*0.7 = 17.5 ptos

Distancia:
Super. Y -- 1025m => 100 - 100 = 0 ptos => 0*0.5 = 0 ptos
Super. X -- 725m => 100 - 70.73 = 29.27 ptos => 29.27*0.5 = 14.635 ptos
Super. W -- 585m => 100 - 57.07 = 42.93 ptos => 42.93*0.5 = 21.465 ptos
Super. Z -- 300m => 100 - 29.26 = 70.74 ptos => 70.74*0.5 = 35.37 ptos

Calidad:
Super. W -- 5 => 100*0.7 = 70 ptos
Super. X -- 3.5 => 70*0.7 = 49 ptos
Super. Y -- 3 => 60*0.7 = 42 ptos
Super. Z -- 1 => 20*0.7 = 14 ptos

Ahora se realizará sumatorio total de cada centro:

Sumatorio total:
Super. X => 1.596 + 14.635 + 49 = 65.231 ptos
Super. Y => 17.5 + 0 + 42 = 59.5 ptos
Super. W => 0 + 21.465 + 70 = 91.465 ptos
Super. Z => 3.115 + 35.37 + 14 = 52.465 ptos

Y finalmente el sistema muestra los 3 mejores centros que tienen más puntuación en el orden que corresponde, es decir:

1. **Super. W → 91.465 ptos**
2. **Super. X → 65.231 ptos**
3. **Super. Y → 59.5 ptos**

5. Conclusiones

5.1. Planificación del proyecto

En este apartado explicaré las tareas que en un principio estaban establecidas y veremos como la aplicación ha ido sufriendo una evolución a lo largo del tiempo añadiendo alguna funcionalidad y modificando los tiempos de desarrollo de la tarea.

En un principio este proyecto estaba pensado finalizarlo el cuatrimestre pasado (Cuatrimestre otoño 2011 – 2012). A pesar de estas previsiones por motivos personales tuve que centrarme en otros asuntos y no me fue posible desarrollar la aplicación.

El primer diagrama de Gantt (*Figura 5.1*) que creé no incluye alguna de las tareas que más adelante vi factibles de hacer:

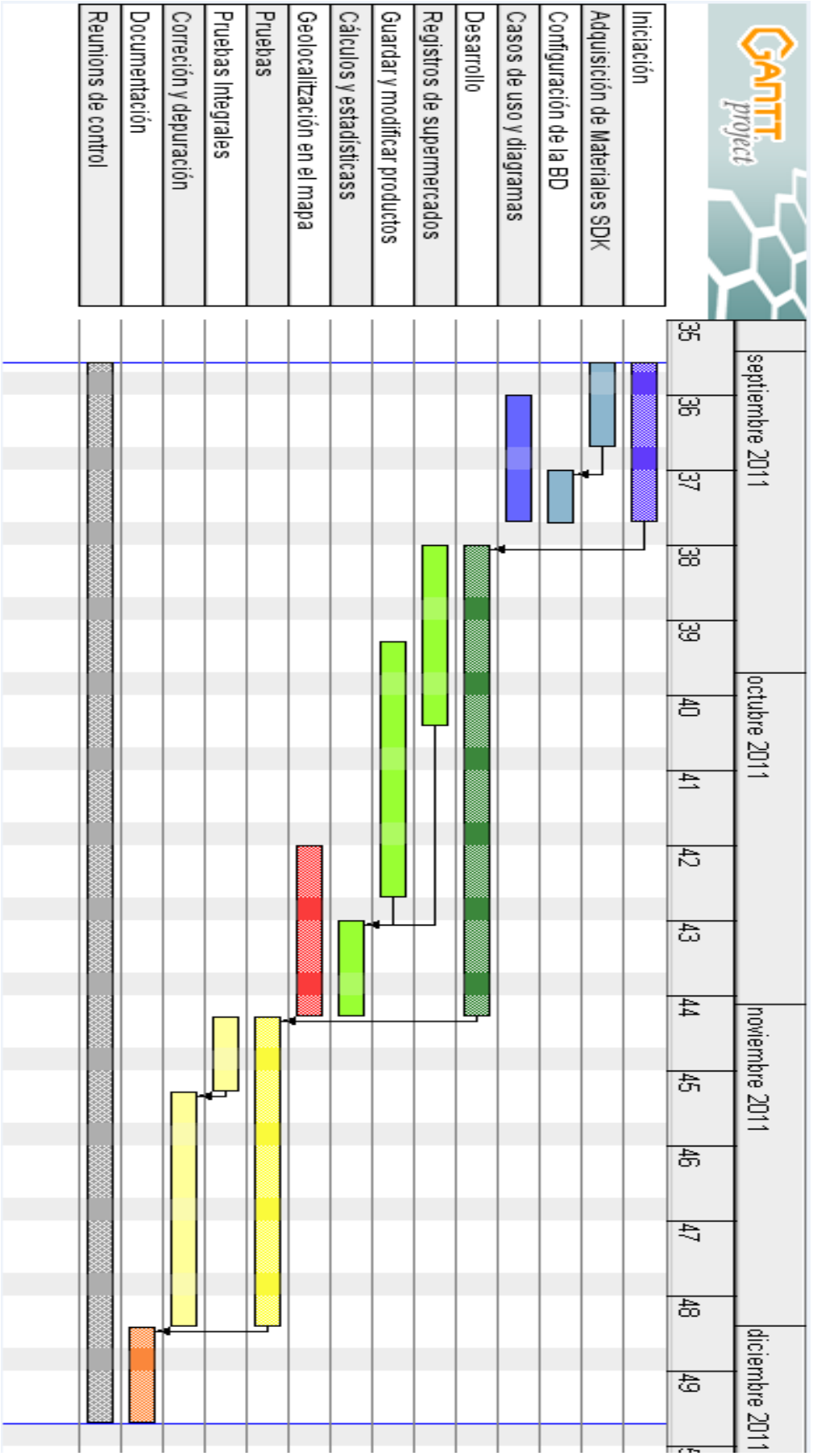


Figura 5.1 – Diagrama de Gantt inicial

Actualmente trabajando a media jornada y cursando otros estudios extrauniversitarios la tarea se ha vuelto más sencilla de realizar.

La nueva planificación (*Figura 5.2*) contiene la funcionalidad de conexión vía internet para actualizar la base de datos del sistema. Además creí conveniente dedicarme cierto tiempo a diseñar una capa de presentación mejorada. Estas dos tareas hicieron que tuviese que reorganizar el tiempo quedando de esta manera:

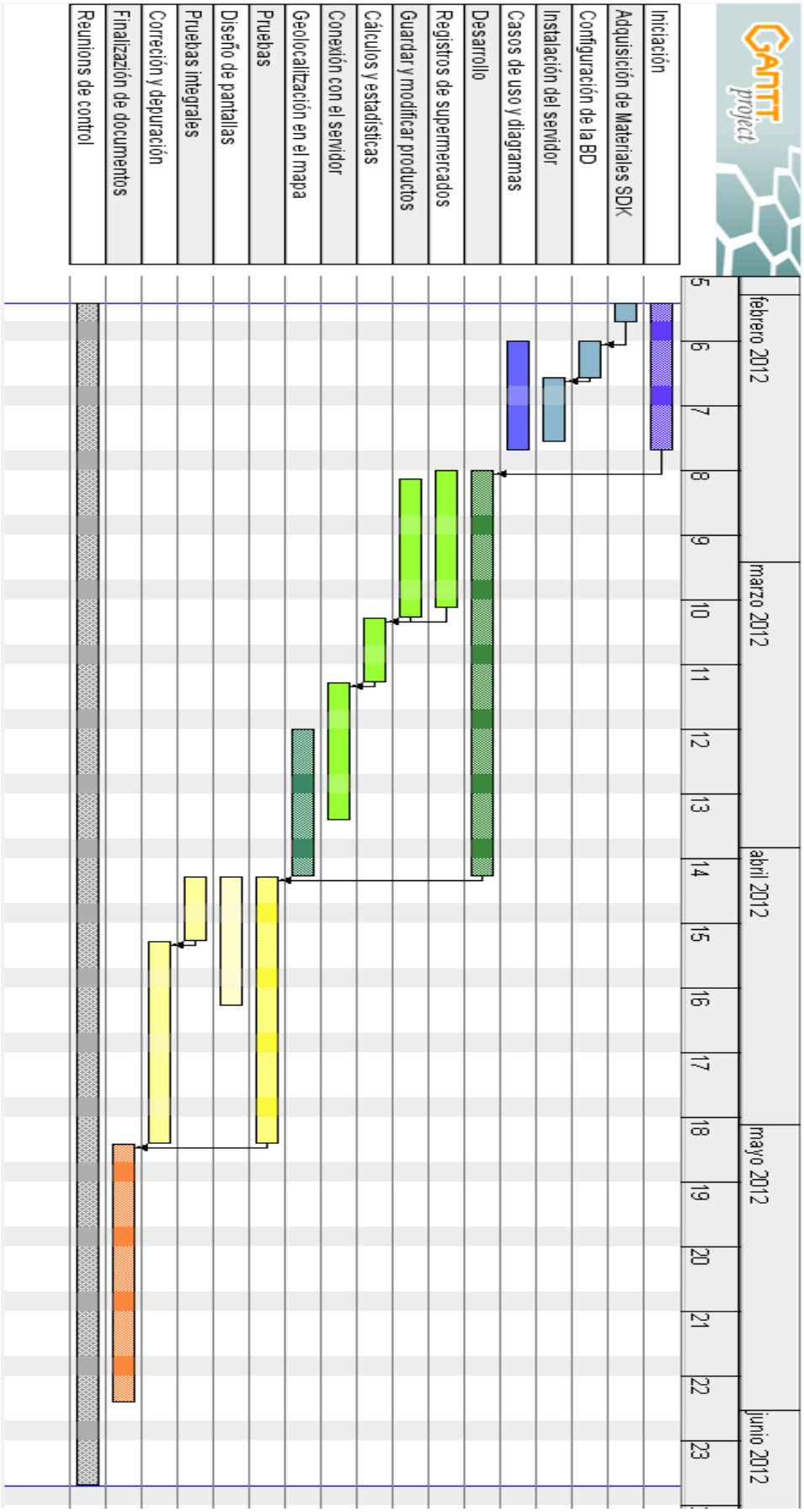


Figura 5.2 – Diagrama de Gantt final

La planificación del proyecto la podemos dividir en 4 partes principales:

- **Iniciación**
- **Desarrollo**
- **Pruebas**
- **Documentación**

La parte de iniciación fue un trabajo no tan duro este cuatrimestre ya que al estar pensada esta aplicación para el curso anterior tuve cierto tiempo para formarme y configurar ciertas cosas. Eso me proporciono un cierto colchón de tiempo “extra” para aplicar un margen de contingencia en las tareas futuras que pudiesen tener mas dificultad. Esta parte conllevó 17 días de trabajo en total.

La parte de desarrollo duró 60 días. Como se puede observar esta fase es la más extensa ya que tuve que investigar posibles problemas y dudas que surgían a lo largo del proceso. Además me fue necesario desarrollar dos tareas a la vez si quería finalizar el proceso ya que algunas condicionaban el poder empezar otras. Este es el caso de realizar cálculos para la compra. Mientras no pudiese tener la gestión de los productos y los supermercados bien implementados no podría realizar los cálculos de compra correctamente.

En la fase de pruebas fue muy importante realizar todos los testeos posibles para intentar después ponerme completamente con la implementación. La fase en global duró 30 días.

La documentación me llevó 25 días ya que una vez echa luego tuve que ir modificando correcciones que el tutor y yo creíamos convenientes realizar.

No cabe olvidar que a lo largo del proyecto se han ido realizando reuniones de control para comprobar la situación actual de este y solventar posibles dudas que tenia.

En resumen el proyecto ha supuesto 128 días de trabajo globales. Si tenemos en cuenta que cada día le he dedicado 4 horas el resultado final de horas trabajadas es 512 horas.

5.2. Estudio económico

Después de haber finalizado el diseño de la metodología, y una vez ya la tenemos aplicada al proyecto de EasyMarket, queda por calcular el coste económico que tendría el desarrollo de un proyecto con estas magnitudes si una empresa decidiese hacerlo.

La *Tabla 5.1* muestra un resumen del coste temporal de todos los casos expuestos en el Gantt final que he mostrado en el apartado previo.

Tarea	Nº Horas	Inicio	Fin
Adquisición de materiales SDK	2	02/02/2012	04/02/2012
Configuración de la BD	4	06/02/2012	10/02/2012
Instalación del servidor	7	10/02/2012	17/02/2012
Casos de uso y diagramas	12	06/02/2012	18/02/2012
Registro de Supermercados	15	20/02/2012	06/03/2012
Guardar y modificar productos	15	21/02/2012	07/03/2012
Cálculos y estadísticas	7	07/03/2012	14/03/2012
Conexión con el servidor	15	14/03/2012	29/03/2012
Geolocalización con el mapa	16	19/03/2012	04/04/2012
Diseño de pantallas	14	04/04/2012	18/04/2012
Pruebas integrales	7	04/04/2012	11/04/2012
Corrección y depuración	22	11/04/2012	03/05/2012
Finalización de documentos	28	03/05/2012	31/05/2012
Reuniones de control	128	02/02/2012	09/06/2012

Figura 5.3 – Tabla de coste temporal

Si partimos de la base que la empresa tendrá un director de proyecto, un diseñador, un programador y un arquitecto repartiríamos las horas de la siguiente manera atribuyendo una remuneración justa para cada trabajador:

Puesto	Euros/Hora	Horas de trabajo	Sueldo final
Director de proyecto	60 €	56 h	3360 €
Arquitecto	50 €	48 h	2400 €
Programador	30 €	408 h	12240 €
Diseñador	25 €	56 h	1400 €
TOTAL	-	-	**19400 €**

Figura 5.4 – Tabla de presupuesto económico

Por tanto el coste final del proyecto dentro de una empresa seria de 19.400 € aproximadamente.

5.3. Trabajo futuro

EasyMarket es una aplicación que podría servir en el futuro para otros objetivos además de los ya creados. Ahora mismo es una aplicación que funciona perfectamente. Sin embargo siempre hay aspectos que se pueden mejorar.

En primer lugar el tema UI podría ser más atractivo para el cliente si alguien especializado en este tema lo profundizase más. Yo al no tener conocimientos de diseño visual quizá no he podido explotar todas las posibilidades que existen para atraer la atención de los usuarios. En el tema del sonido también tendría repercusión a las personas que se dedican a la parte grafica ya que yo de momento he usado sonidos libres para no evitar gastos indebidos en el proyecto.

Otra parte que sería conveniente mejorar es la parte de actualización. El sistema actualiza todos los datos por medio de una ip fijada desde la aplicación. Sin embargo, las ip que asignan los Proveedores de Servicio a Internet (ISP)

suelen ser dinámicas a no ser que paguemos por una fija. Por eso el problema que existe actualmente en este ámbito es que en el momento que mi proveedor cambie la dirección ip de mi servidor, el dispositivo no podrá conectarse vía internet. Para solucionar este problema existen compañías externas las cual se encargan de dar soluciones de DNS a IPs dinámicas. Estas empresas establecidas en la nube se encargan de redireccionar la IP y ofrecen a particulares la posibilidad de crear un servidor en Internet con una simple dirección la cual siempre contiene la IP actual del router que tengamos configurado (un ejemplo de este servicio es DynDNS). Al principio de desarrollar esta funcionalidad la aplicación estaba desarrollada de esta manera: el dispositivo se conectaba a un servidor en la nube el cual estaba conectado con mi propio pc servidor. Al hacer la consulta le retornaba la ip actual a la cual debía conectarse y así siempre tenia la ip a la cual debía hacer las peticiones de transferencia. Sin embargo a principios del 2012 este servicio dejó de funcionar gratuitamente y no he logrado otro que pueda hacerlo sin ánimo de lucro.

Una funcionalidad no implementa y que podría ser interesante para mejorar la usabilidad es ver los precios de un producto en los diferentes supermercados con solo leer el código de barras de este a través de la cámara. Por ejemplo, si una persona se encuentra en su casa y quiere consultar el precio de tetrabrik de leche que tiene, con solo poner el código de barras en la cámara el sistema le daría una lista de los centros que lo tienen con sus respectivos precios.

Otra funcionalidad que se podría implementar seria ver una grafica de cada centro mostrando la evolución de calidad que ha tenido a lo largo del tiempo. Cogiendo las valoraciones que ha recibido podríamos englobarlas cada 6 meses y observar si esa empresa ha mejorado su calidad o por lo contrario ha empeorado.

Si quisiéramos tener algún beneficio con esta aplicación podríamos establecer convenios con empresas para que muestren banners de publicidad. Además, comentando el tema de beneficios económicos, se podría tener un convenio con las empresas que permiten la compra vía online y así establecer un contrato con ellos para poder implementar dicha opción en el programa. Otra forma de beneficiarse seria tener dos tipos de aplicaciones: la gratuita y la de pago. La

gratuita ofrecería menos cantidad de productos, centros, ubicaciones, etc. Y la opción de pago tendría todas las funcionalidades y datos que existen.

5.4. Conclusiones personales

Durante el proyecto he aprendido a utilizar lenguajes de programación de Android y sus diversas APIs. El trabajo para llevar a cabo todos los objetivos ha sido muy exhaustivo ya que mis conocimientos en este ámbito eran nulos.

También he tenido la oportunidad de ampliar mis conocimientos y la experiencia que tenia en el proceso de desarrollo de un proyecto. Además he podido aplicar gran cantidad de conocimientos que he adquirido a lo largo de la carrera como son el diseño de las estructuras de datos, la orientación a objetos, el uso de las bases de datos, uso de herramientas de programación, conexiones servidores – cliente, desarrollo de documentación, etc.

He intentado que el programa posea ciertas características:

- **Multiplataforma:** Pretendo que la aplicación sea lo mas sencilla de exportar a otros SOs como puede ser OSX. Por eso mismo la base de datos en este caso recoge toda la cantidad de información ya que SQLite es un modelo de gestión de datos que usan múltiples plataformas.
- **Usable:** Lo que pretendía es que EasyMarket fuese muy cómodo de usar y el usuario se sintiera cómodo usándolo.
- **Eficiente:** La rapidez de respuesta en una aplicación es muy importante tanto en las aplicaciones móviles como el software de ordenadores.
- **A prueba de errores:** He realizado la depuración y los juegos de prueba intentando probar todos los casos posibles para evitar errores.
- **Sencillo:** La intención de la aplicación es que sea fácil de usar y con pocas funcionalidades pero que sea efectivas y sin problemas. Un programa muy complejo para una aplicación móvil puede resultar muy pesado e incomodo a la hora de utilizarlo.

- **Integrado con Internet (o la nube):** Actualmente la gran mayoría de smartphones están conectados a internet y el uso de compartición de información es muy común. La gente puede tener una motivación extra al pensar que los datos que comentan o valoran desde su dispositivo pueden ser visibles para ayudar a otros usuarios con su opinión.

Finalmente podemos decir que, con estos objetivos cumplidos, hemos alcanzado la meta de la aplicación con éxito.

6. Bibliografía

En la parte de bibliografía mostraré todas fuentes de información a través de las cuales he extraído o consultado algún tipo de información para desarrollar mi proyecto.

6.1. Artículos de consulta

Evolución SmartPhones - *http://blog.nielsen.com/nielsenwire/online_mobile/android-most-popular-operating-system-in-u-s-among-recent-smartphone-buyers/*

Estadística de mercado de móviles - *http://www.gartner.com/it/page.jsp?id=1924314*

Estadística de mercado de móviles 2 - *http://www.todoapps.net/2012/05/estadisticas-de-android-actualmente-en.html*

La evolución de Android - *http://tecnologia.elpais.com/tecnologia/2012/02/27/actualidad/1330361316_288568.html*

Evolución de malware en Android - *http://www.securitybydefault.com/2011/02/evolucion-del-malware-en-dispositivos.html*

Arquitectura Android - *http://commons.wikimedia.org/wiki/File:System-architecture.jpg?uselang=es*

Foro consultas - *http://www.android-spa.com*

Foro consultas 2 - *http://stackoverflow.com/*

6.2. Manuales y documentos de estandarización

Sockets en Java - *http://www.dlsi.ua.es/asignaturas/sid/JSockets.pdf*

Google Maps en Android - *http://www.slideshare.net/oscarsalguero/usando-google-maps-en-tu-android-app*

Manual para programar Android - *http://www.sgoliver.net/blog/*

Manual para programar Android 2 - *http://www.elbauldelprogramador.com/programacion/*

ListAdapter de Android - *http://es.scribd.com/doc/61548244/7/ListAdapter-ArrayAdapter-SpinnerAdapter-SimpleCursorAdapter*

Tutorial configuración Android - *http://yosoyandroid.com/2011/05/mis-primeros-pasos-en-programacion-android-mi-primer-hello-world/*

Guía Android - *http://developer.android.com/guide*

7. Agradecimientos

Primeramente quera dar las gracias a mi tutor el cual sin él no habría podido comenzar este tipo de proyecto ya que la idea inicial fue suya.

Además quería agradecer a todas esas personas; amigos, familia, pareja que han estado ahí para darme ánimos y motivación para continuar.

Por ultimo, muchas gracias a esos lugares de información de Android los cuales me han podido ayudar a resolver todos mis problemas tanto de forma directa como indirecta.