ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DE COMPUTADORES

# APLICACIÓN SOCIAL ANDROID PARA LA BÚSQUEDA DE PRECIOS DE PRODUCTOS POR LOCALIZACIÓN

# SOCIAL ANDROID APPLICATION FOR SEARCHING PRODUCTS PRICES BY LOCATION

Realizado por
**Pedro Gámez García**
Tutorizado por
**Daniel Garrido Márquez**
Departamento
**Lenguajes y ciencias de la computación**

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Noviembre de 2014

Fecha defensa:
El Secretario del Tribunal

**Resumen:** En este trabajo se ha desarrollado una aplicación para dispositivos que utilizan el sistema operativo Android. La aplicación desarrollada sirve para intentar localizar los mejores precios de determinados productos en tiendas físicas dentro de una zona determinada, que podrá ser seleccionada por el propio usuario.

Se le ha dado un enfoque puramente social, de manera que puedan ser los propios usuarios de la aplicación quienes introduzcan los contenidos de la misma. Por ejemplo, un usuario de la aplicación ve una buena oferta de un producto y, con solo añadirlo a la aplicación, lo estará compartiendo con el resto de usuarios. Otro usuario puede ver algunos detalles de la oferta introducida como, por ejemplo, su precio, el nombre del comercio que la ofrece, la dirección del comercio, verla en Google Maps, votarla positiva o negativamente o añadir comentarios que podrán ver todos los usuarios.

Además, se ha implementado un perfil del usuario donde éstos pueden gestionar sus datos personales, su ubicación de búsqueda y sus publicaciones. El usuario podrá editar o eliminar cualquier producto u oferta que haya creado él mismo.

También se ha añadido un sistema de usuarios administradores que tendrán total control sobre la aplicación y podrán editar o borrar las publicaciones de otros usuarios en caso de que sea necesario.

**Palabras claves:** Android, Google App Engine, social, compartir, producto, oferta, precio, localización, Google Maps.

**Abstract:** This project has developed an application for devices using the Android operating system. This application allows you to try locating the best prices of certain products in physical stores in a certain area, which can be selected by the user.

The application has a purely social approach, so the users of the application themselves will be the ones filling its content. For example, an application user sees a good offer on a product and by adding it to the application, that offer will be shared with the rest of the users. Another user can see some details about the shared offer like, for example, the price, the shop name, the address, see it on Google Maps, vote it positive or negatively or add comments which can be seen for the rest of the users.

In addition, an user profile has been implemented where the users can manage their personal data, the search location and their shares. The user can edit or delete any product or offer if it has been created by himself.

A system that includes users administrators has also been added and they will have total control over the application so they can edit or delete other users shares if it is necessary.

**Keywords:** Android, Google App Engine, social, share, product, offer, price, location, Google Maps.

# Índice

# 1. Introducción

## 1.1. Motivación del TFG

La idea para la realización del presente *TFG* surge ante la pregunta que en numerosas ocasiones suele surgirnos cuando vamos a comprar un determinado producto en una tienda física: ¿es la mejor opción o estará más barato en otro sitio al que pueda ir?. Si en el momento en el que nos surge esa pregunta, sin importar dónde nos encontremos, podemos consultar el mejor precio dentro una zona determinada, ahorraríamos dinero y tiempo, ya que a veces hay que visitar varias tiendas para comparar los precios que tienen en ese momento y que, además, suelen ser muy variables.

Consultar esta información debería ser posible utilizando algo que la mayoría de la gente suela llevar siempre encima, por lo tanto, un dispositivo móvil, como puede ser un teléfono, es la mejor opción. Dentro de este tipo de dispositivos hay varias opciones en cuanto al Sistema operativo sobre el que se podría desarrollar una aplicación de esas características. En este caso, hemos decidido utilizar el sistema operativo *Android* debido a que es uno de los sistemas operativos más utilizados con diferencia en todo el mundo.

Pensamos que es fundamental que la aplicación sea de carácter social, que sea la propia gente quien introduzca sus contenidos y comparta opiniones con el resto de usuarios.

## 1.2. Objetivos del TFG

El objetivo fundamental del *TFG* es el desarrollo de una aplicación social para dispositivos con sistema operativo *Android* que permita a sus usuarios realizar una búsqueda por localización de determinados productos para comparar los precios que ofrecen las tiendas físicas situadas en una zona especificada por el propio usuario. Objetivos del desarrollo:

- Diseño e implementación de una interfaz de usuario que sea atractiva, cómoda y fácil de utilizar.
- Se utilizará *Google APP Engine*, que permite el uso y manteniento de una base de datos en la nube, para el desarrollo de la base de datos que almacenará la información acerca de los usuarios y la que ellos mismos introduzcan.
- Acceso a *Google Maps* desde la propia aplicación para mostrar en un mapa la localización de las ofertas introducidas por los usuarios.
- Acceso a *GPS* y a internet para dar la opción al usuario de introducir la dirección de una oferta con sólo pulsar un botón, facilitando así el proceso de añadir ofertas a la base de datos.

- Búsqueda de productos y ofertas teniendo en cuenta el nivel de búsqueda que desee hacer el usuario: local, provincial o nacional.
- Sistema de acceso a la aplicación, registro de usuarios y envío de recordatorio de datos de acceso al correo electrónico del usuario.
- Introducción, por parte del usuario, de productos, ofertas de los productos, votos y comentarios.
- Introducción de imágenes de los productos y subida a la base de datos. Las imágenes podrán ser seleccionadas desde la galería del dispositivo o directamente haciéndole una foto al producto.
- Sistema de votos y comentarios sobre las ofertas introducidos por los propios usuarios. Esto que permitirá el intercambio de información acerca de las ofertas entre todos los usuarios de la comunidad.
- Panel de control de usuario donde pueda gestionar sus datos personales, publicaciones, localización de búsqueda, etc.

## 1.3. Estructura de la memoria

A continuación se realizará una breve descripción de los capítulos que componen el presente volumen.

### *Capítulo 2. Tecnología utilizada*

En este capítulo se explicará la tecnología utilizada para desarrollar la aplicación objetivo del presente trabajo.

### *Capítulo 3. Análisis y especificación*

En este capítulo se exponen los requisitos funcionales y no funcionales que debe cumplir la aplicación a desarrollar. Además, se muestra el diagrama de casos de uso y su descripción textual.

### *Capítulo 4. Diseño e implementación*

Se trata del capítulo en el que se muestran los detalles más importantes sobre la implementación de las distintas partes que componen este trabajo. El capítulo se divide en dos partes bien diferenciadas, la primera parte dedicada a la base de datos y la segunda a la aplicación en sí.

### *Capítulo 5. Pruebas*

En el quinto capítulo se explica el procedimiento utilizado para la ejecución de las pruebas y se muestra un ejemplo en detalle.

### *Capítulo 6. Conclusiones y líneas futuras*

Este capítulo está destinado a sacar conclusiones acerca del trabajo desarrollado y a establecer unas posibles líneas futuras a seguir para el desarrollo y mejora de las funciones ofrecidas por la aplicación.

### *Anexos técnicos*

Se adjunta el manual de usuario de la aplicación.

# 2. Tecnología utilizada

## 2.1. Sistema Operativo Android

*Android* es un sistema operativo basado en el *kernel* de *Linux* diseñado principalmente para dispositivos móviles con pantalla táctil como teléfonos inteligentes o tabletas. Actualmente, se trata de uno de los sistemas operativos más conocidos y utilizados en todo el mundo.

## 2.2. Google APP Engine

Es una plataforma como servicio (*PaaS*) que permite ejecutar aplicaciones en la infraestructura de *Google*. Este servicio de alojamiento web es gratuito hasta determinadas cuotas. Si no se cuenta con un dominio propio, *Google* proporciona uno de la forma *midominio.appspot.com*. Actualmente, las cuentas gratuitas tienen un límite de 500 megabytes de almacenamiento permanente y la suficiente cantidad de ancho de banda y CPU para cinco millones de visitas mensuales, y si la aplicación supera estas cuotas, se pueden comprar cuotas adicionales. Las aplicaciones que utilizan *Google App Engine* se pueden implementar mediante los lenguajes de programación *Python*, *Java*, *Go* y *PHP*. El presente trabajo será desarrollado utilizando *Java* como lenguaje de programación.

*Google App Engine*, además, cuenta con la posibilidad de utilizar un servidor local que simula el funcionamiento del servidor remoto. Esto permite poder probar las aplicaciones con mayor seguridad ya que, si ocurre cualquier error, basta con restaurar el fichero que almacena la base de datos local y volver a empezar. Además, se evita enviar tráfico al servidor remoto a través de internet.

Otro aspecto importante es la existencia de una consola de administración, tanto local como remota, que permite al desarrollador tener control sobre el estado de la aplicación y realizar tareas de mantenimiento de una manera sencilla e intuitiva. La *figura 1* muestra el aspecto de la consola remota de administración de *Google App Engine*.

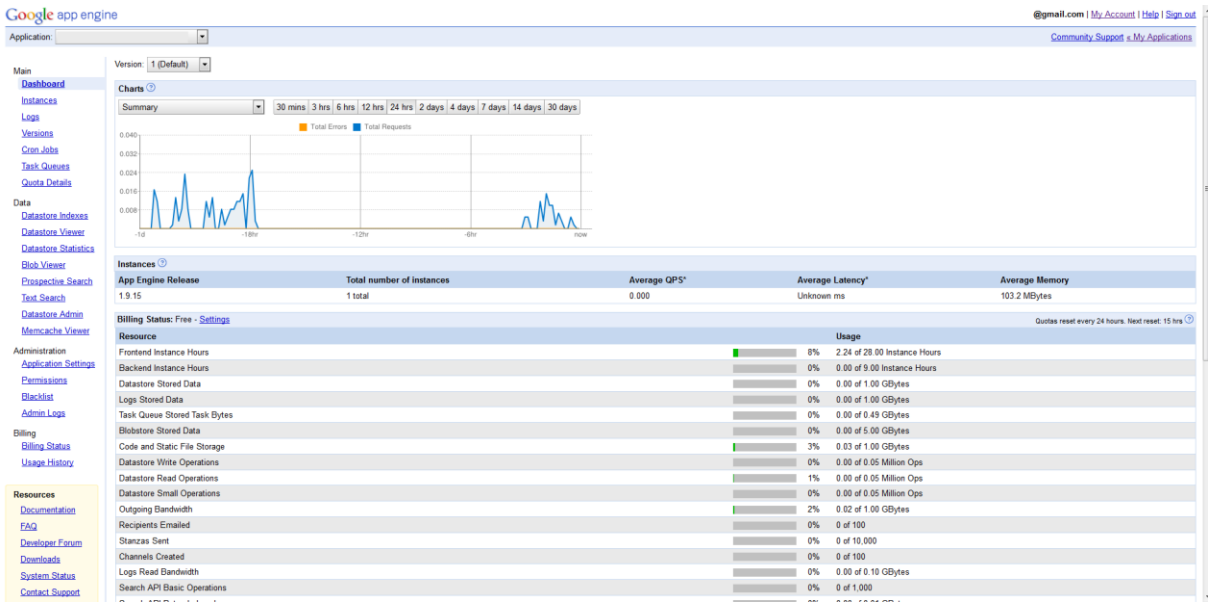


Figura 1. Consola de administración de Google App Engine

## 2.2.1. Almacenamiento de datos

El entorno de *Google App Engine* proporciona tres opciones para almacenar los datos de la aplicación: *App Engine Datastore*, *Google Cloud SQL* y *Google Cloud Storage*. Nosotros nos decantamos por *App Engine Datastore*, básicamente porque es la única opción gratuita de las tres y también la más extendida.

*App Engine Datastore* es un almacén de datos sin esquema, *NoSQL*, que proporciona almacenamiento robusto y escalable para las aplicaciones web. Este almacenamiento no es de tipo relacional, sino que es distribuido y orientado a objetos, en el que cada entidad del *Datastore* tiene un identificador asociado. El *Datastore* almacena objetos conocidos como entidades. Una entidad tiene una o más propiedades, valores con nombre de una gran cantidad de tipos de datos soportados. Se pueden realizar múltiples operaciones en una única transacción, que no se considerará satisfactoria a no ser que todas las operaciones que incluye sí lo sean. Esto es especialmente útil cuando muchos usuarios pueden acceder al mismo tiempo y manipular los mismos datos a la vez.

El *Java Datastore SDK* incluye implementaciones de las interfaces *Java Data Objects* (*JDO*) y *Java Persistence API* (*JPA*), así como la *Datastore API* de más bajo nivel. Para simplificar el uso del *Datastore*, existen conjuntos de librerías tales como *Objectify* o *Slim3*. Utilizaremos *Objectify* por ser muy simple de utilizar y porque evita muchas complejidades presentadas por *JDO/JPA* y el tratamiento a bajo nivel del *Datastore*.

## 2.3. Entorno de programación Eclipse

*Eclipse* es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar software.


Eclipse facilita considerablemente el desarrollo de aplicaciones para *Android* ya que automatiza muchas tareas que tendrían que ser realizadas a nivel de comandos de consola, ahorrando mucho tiempo al desarrollador. Se utilizará esta herramienta para desarrollar, utilizando el lenguaje de programación *Java*, todo el software del presente trabajo.

### 2.3.1. Android SDK

Para poder programar con *Android* utilizando *Eclipse* es necesario instalar *Android SDK*, que proporciona las librerías *API* y las herramientas de desarrollo necesarias para compilar, probar y depurar aplicaciones para *Android*.

### 2.3.2. Android Development Tools (ADT)

Otro elemento necesario es *Android Development Tools* (*ADT*), que es un *plugin* para *Eclipse* diseñado para proporcionar un entorno adecuado para el desarrollo de aplicaciones *Android*. Mediante este *plugin* se pueden crear rápida y fácilmente proyectos *Android*, añadir paquetes basados en el *Android Framework API*, depurar aplicaciones utilizando un emulador de un dispositivo *Android* (*AVD*) y exportar aplicaciones firmadas o sin firmar en formato .apk.

Se pueden emular dispositivos *Android* seleccionando opciones muy específicas como la versión de *Android* (*API*) que tendrá instalada el dispositivo, el microprocesador (*ARM*, *Intel…*), memoria *RAM*, tarjeta *SD*, emulación de cámara, etc.


Figura 2. Android Virtual Device (AVD)

### 2.3.3. Google Plugin for Eclipse

Es un conjunto de herramientas para el desarrollo de software que facilita el diseño, construcción, optimización y alojamiento de aplicaciones basadas en la nube.

Este *plugin* permite:

- Crear y alojar aplicaciones *Google App Engine*.
- Utilizar *Cloud SQL* como base de datos para la aplicación *Google App Engine*.
- Importar las últimas *APIs* de *Google*.
- Utilización de *APP Engine* como backend para los proyectos *Android*.
- Soporte para *Cloud Endpoints*, que permite un acceso más sencillo del usuario a los *App Engine backends*.
- Alojar las aplicaciones con sólo un clic.
- Proyectos *App Engine* basados en *WTP*.

### 2.3.4. Google Maps API

*Google Maps API v2* permite al usuario explorar el mundo utilizando los mapas de *Google*. El usuario no sólo podrá navegar por el mapa, si no también añadir marcadores, dibujar líneas y polígonos para marcar caminos o regiones, utilizar el zoom, rotar la cámara, calcular distancias, etc. Esta *API* requiere que se establezca la versión mínima de compatibilidad con sistemas operativos *Android* a la versión *API 14*.

Figura 3. Google Maps API

### 2.3.5. JavaMail

La *API* de *JavaMail* proporciona una plataforma independiente y un conjunto de librerías de protocolo independiente para construir aplicaciones de mensajería y correos electrónicos.

Estas librerías incluyen clases que modelan el sistema de envío de correos electrónicos. Entre otras cosas, permite especificar el correo electrónico de entrada y salida, el servidor de salida, el asunto del mensaje, el cuerpo, etc.

### 2.3.6. Objectify API

*Objectify* es una *API* para el acceso a datos en *Java* diseñada para *Google App Engine Datastore*. Sus objetivos son:

- Ser fácil de aprender y entender.
- Soportar todas las funciones nativas del *Datastore* de manera intuitiva.
- Modelo sofisticado, fuerte tipificación, estructuras de datos polimórficas.
- Habilita lógica transaccional tipo *EJB* modular.
- Aumenta el rendimiento y reduce el coste a través de un inteligente uso de la caché en las transacciones.
- Permite realizar migraciones de esquemas al momento sin tener que parar el servidor.
- Puede coexistir con otras herramientas del *Datastore*: el *API* de *Java* de bajo nivel, *JDO/JPA*, *Twig*, *Go*, *Python DB* y *NDB*.

*Objectify* pretende buscar un nivel de abstracción adecuado, ni demasiado alto, ni demasiado bajo.

# 3. Análisis y especificación

## 3.1. Requisitos

Después de realizar un estudio sobre las funciones que debe cumplir la aplicación para cumplir con los objetivos fijados, se han establecido una serie de requisitos funcionales y no funcionales que serán expuestos a continuación.

### 3.1.1. Requisitos funcionales

Todos los requisitos funcionales que se enumeran a continuación deben poder ser realizados desde la propia interfaz gráfica de la aplicación.

1. **Crear usuarios.** Se deben poder añadir nuevos usuarios.
2. **Recuperar datos de acceso**. El usuario debe tener la posibilidad de recuperar sus datos de acceso, que serán enviados a su correo electrónico si así lo solicita.
3. **Modificación de datos de acceso**. El usuario debe poder modificar sus datos de acceso. Como mínimo, se le permitirá la modificación de su contraseña y correo electrónico.
4. **Acceso a la aplicación**. Es necesario crear una pantalla de acceso a la aplicación para verificar que el usuario existe en la base de datos y su contraseña es correcta.
5. **Añadir un producto**. Se deben poder añadir nuevos productos a la aplicación.
6. **Modificar un producto**. El usuario debe poder modificar cualquier información sobre un producto que él mismo haya introducido. Un usuario administrador podrá modificar cualquier producto. Esta modificación incluye también la eliminación del producto.
7. **Añadir una oferta**. Se deben poder añadir ofertas, que estarán relacionadas con los productos existentes o con uno que se esté introduciendo en el mismo momento.
8. **Modificar una oferta**. El usuario debe poder modificar cualquier información sobre una oferta que él mismo ha introducido. Un usuario administrador podrá modificar cualquier oferta. Esta modificación incluye también la eliminación de la oferta.
9. **Buscar ofertas**. El usuario podrá buscar ofertas del producto que desee.
10. **Ver la información completa de una oferta.** El usuario debe poder acceder a la información de las ofertas.
11. **Sistema de votación.** Implementar un sistema de votos en el que el usuario vote positiva o negativamente una oferta. Puede servir no sólo para dar una noción de lo buena o mala que es una oferta, sino también para informar sobre su veracidad.

12. **Sistema de comentarios sobre las ofertas.** Implementar un sistema de comentarios donde los usuarios puedan expresar sus opiniones sobre las ofertas.
13. **Ver la dirección del comercio en un mapa.** Se le debe dar la posibilidad al usuario de ver en un mapa dónde se encuentra el comercio que ofrece un determinado producto.
14. **Establecer ubicación de búsqueda.** El usuario debe poder establecer en qué zona desea buscar las ofertas de los productos.
15. **Desconexión.** Se podrá desconectar en cualquier momento de la aplicación, volviendo a la pantalla de acceso.
16. **Salir de la aplicación.** Se debe poder salir de la aplicación.
17. **Recordar datos de acceso al reiniciar.** Se deben poder almacenar los datos de acceso de un usuario, así como cualquier otra información relevante, para poder recuperarlos en cada ejecución de la aplicación en caso de ser necesario.

### 3.1.2. Requisitos no funcionales

La aplicación a desarrollar tiene que cumplir una serie de requisitos de rendimiento, diseño, facilidad de uso, etc. A continuación se describen los requisitos no funcionales de la aplicación a desarrollar.

1. **Desarrollar la aplicación para el sistema operativo Android**. Se debe desarrollar una aplicación compatible con dispositivos que utilicen este sistema operativo.
2. **Interfaz gráfica sencilla e intuitiva**. Se debe proporcionar una interfaz de usuario que sea fácil de utilizar.
3. **Automatización de tareas**. Se debe intentar automatizar al máximo las tareas para facilitar al usuario la manipulación de la aplicación, sobretodo en la búsqueda e introducción de productos y ofertas.
4. **Conectividad.** La aplicación ha de ser capaz de comunicarse de manera fluida con el servidor.
5. **Eficiencia.** La aplicación debe ser eficiente, independientemente de los posibles retardos que pueda producir el servidor remoto.
6. **Robustez.** Hay que intentar depurar al máximo todos los errores que vayan surgiendo.
7. **Extensibilidad.** Se debe dejar la aplicación lo mejor preparada posible para la posible incorporación de nuevas funciones en el futuro.
8. **Documentación.** Se debe contar con un manual de usuario en el que se explique el funcionamiento y manejo de la aplicación.

## 3.2. Casos de uso

Antes de comenzar el desarrollo de la aplicación es preciso establecer cuáles serán las funciones que la aplicación deberá poder realizar para poder cumplir con los objetivos del trabajo. Los casos de uso son un elemento importante para ver de

manera rápida e ilustrativa lo que va a poder hacer nuestra aplicación. La figura 4 muestra el diagrama de casos de uso.

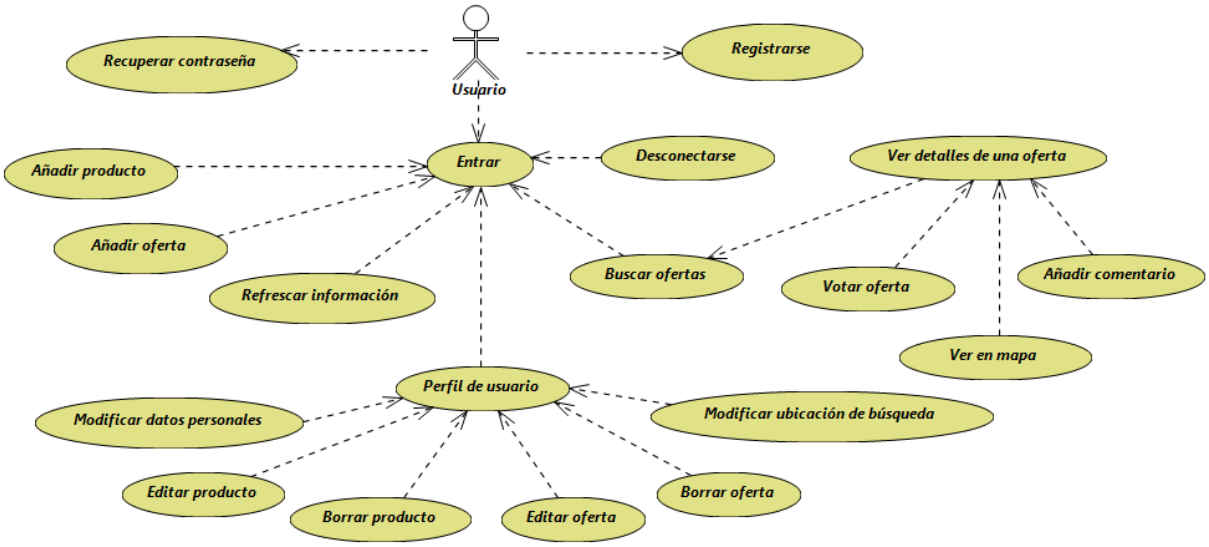


Figura 4. Diagrama de casos de uso

A continuación se realiza una descripción de los casos de uso que aparecen en la figura 4.

1. **Caso de uso**: Registrarse.
   **Actor principal**: Usuario.
   **Pre-condiciones**: La aplicación debe estar en ejecución, con acceso a internet y mostrando la pantalla de acceso. La aplicación sólo podrá ser ejecutada en dispositivos *Android* a partir de la *API* mínima soportada (*API 14*).
   **Escenario de éxito principal**:
   1. El usuario pulsa el botón "Registrar".
   2. Rellena los campos con su información: Nombre de usuario, contraseña, etc.
   3. Pulsa el botón "Aceptar".

2. **Caso de uso**: Entrar.
   **Actor principal**: Usuario.
   **Pre-condiciones:** La aplicación debe estar en ejecución, con acceso a internet y mostrando la pantalla de acceso. La aplicación sólo podrá ser ejecutada en dispositivos *Android* a partir de la *API* mínima soportada (*API 14*).
   **Escenario de éxito principal**:
   1. El usuario rellena los campos "Nombre de usuario" y "Contraseña" en la pantalla de acceso.
   2. Pulsa el botón "Entrar", el usuario existe en la base de datos y, además, la contraseña que ha introducido coincide con la que tiene el usuario almacenada.

3. **Caso de uso**: Recuperar contraseña.
   **Actor principal**: Usuario.
   **Pre-condiciones**: La aplicación debe estar en ejecución, con acceso a internet y mostrando la pantalla de acceso. La aplicación sólo podrá ser ejecutada en dispositivos *Android* a partir de la *API* mínima soportada (*API 14*).
   **Escenario de éxito principal**:
      1. El usuario pulsa el botón “Recuperar clave”.
      2. Rellena el campo con su correo electrónico, que existe en la base de datos.
      3. La aplicación envía un correo electrónico a la dirección introducida incluyendo los datos de acceso (usuario y contraseña).

4. **Caso de uso**: Buscar ofertas.
   **Actor principal**: Usuario.
   **Pre-condiciones**: El usuario debe haber entrado en la aplicación mediante la verificación de sus datos de acceso y deben existir los productos y ofertas en la base de datos.
   **Escenario de éxito principal**:
      1. El usuario pulsa el botón “Buscar” situado en la barra de acciones superior de *Android*.
      2. Introduce el nombre del producto a buscar. La aplicación le muestra una serie de sugerencias y el usuario puede seleccionar una de ellas si se trata del producto buscado o bien aceptar para que se le muestre una lista de productos con nombre similar o igual al texto introducido.
      3. Si el usuario selecciona una de las sugerencias, pasar al paso número 5, en caso contrario, se mostrará una lista de productos.
      4. El usuario selecciona el producto deseado.
      5. La aplicación muestra todas las ofertas que existen para ese producto dentro de la zona que el usuario tiene configurada en su perfil de usuario.

5. **Caso de uso**: Ver detalles de una oferta.
   **Actor principal**: Usuario.
   **Pre-condiciones**: El usuario debe haber completado el caso de uso número 4.
   **Escenario de éxito principal**:
      1. El usuario selecciona una de las ofertas que se le muestran en pantalla.
      2. La aplicación le muestra todos los detalles de la oferta seleccionada.

6. **Caso de uso**: Ver en mapa
   **Actor principal**: Usuario.
   **Pre-condiciones**: El usuario debe haber completado el caso de uso número 5.
   **Escenario de éxito principal**:
   1. El usuario pulsa el botón "Ver en mapa" en la pantalla que muestra los detalles de la oferta.
   2. La aplicación le muestra un mapa de *Google Maps* con la dirección en la que se encuentra el comercio que ofrece la oferta marcada y enfocada con un zoom adecuado.

7. **Caso de uso**: Votar oferta.
   **Actor principal**: Usuario.
   **Pre-condiciones**: El usuario debe haber completado el caso de uso número 5.
   **Escenario de éxito principal**:
   1. El usuario pulsa el botón "Voto positivo" o "Voto negativo" en la pantalla que muestra los detalles de la oferta.
   2. El usuario confirma el voto positivo o negativo.

8. **Caso de uso**: Añadir comentario.
   **Actor principal**: Usuario.
   **Pre-condiciones**: El usuario debe haber completado el caso de uso número 5.
   **Escenario de éxito principal**:
   1. El usuario pulsa el botón "Comentarios".
   2. La aplicación muestra todos los comentarios de los usuarios para esa oferta.
   3. El usuario pulsa el botón "Añadir" situado en la barra de acciones superior de *Android*.
   4. El usuario introduce el texto del mensaje y pulsa el botón "Aceptar".

9. **Caso de uso**: Añadir producto.
   **Actor principal**: Usuario.
   **Pre-condiciones**: El usuario debe haber entrado en la aplicación mediante la verificación de sus datos de acceso y estar viendo la lista de productos.
   **Escenario de éxito principal**:
   3. El usuario pulsa el botón "Añadir" situado en la barra de acciones superior de *Android*.
   4. Rellena los campos con la información del producto: Nombre del producto, imagen del producto y descripción del producto.
   5. Pulsa el botón "Siguiente".

   **Post-condiciones**: Para poder introducir un producto es necesario añadir también una oferta para ese producto, por lo que se debe proceder a realizar el caso de uso número 10.

10. **Caso de uso**: Añadir oferta.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe haber completado el caso de uso número 9 o bien estar visualizando la lista de ofertas de un determinado producto y pulsar el botón "Añadir" situado en la barra de acciones superior de *Android*.
    **Escenario de éxito principal**:
        1. El usuario rellena los campos con su información: Comercio, dirección, precio, unidades del producto y comentarios acerca de la oferta.
        2. Pulsa el botón "Aceptar".

11. **Caso de uso**: Refrescar información.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe estar situado en cualquier panel que muestre una lista de entidades: productos, ofertas o comentarios de usuarios.
    **Escenario de éxito principal**:
        1. El usuario pulsa el botón "Refrescar" situado en la barra de acciones superior de *Android*.
        2. La aplicación refresca la lista de entidades con los nuevos datos obtenidos de la base de datos.

12. **Caso de uso**: Desconectarse.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe haber completado el caso de uso número 2.
    **Escenario de éxito principal**:
        1. El usuario pulsa el botón "Desconectar" situado en la barra de acciones superior de *Android*.
        2. El usuario confirma que se quiere desconectar y la aplicación le muestra la pantalla inicial de acceso a la aplicación.

13. **Caso de uso**: Perfil de usuario.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe haber entrado en la aplicación mediante la verificación de sus datos de acceso.
    **Escenario de éxito principal**:
        1. El usuario pulsa el botón "Perfil de usuario" situado en la barra de acciones superior de *Android*.
        2. La aplicación le muestra en pantalla su perfil de usuario.

14. **Caso de uso**: Modificar datos personales.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe haber completado el caso de uso número 13.
    **Escenario de éxito principal**:
        1. El usuario rellena todos los campos del apartado “Datos personales”.
        2. El usuario pulsa el botón aplicar para guardar los cambios.

15. **Caso de uso**: Modificar ubicación de búsqueda.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe haber completado el caso de uso número 13.
    **Escenario de éxito principal**:
        1. El usuario rellena todos los campos del apartado “Dónde buscar”, seleccionando la provincia y la localidad.
        2. El usuario selecciona el nivel de búsqueda: local, provincial o nacional.
        3. Pulsa el botón aplicar para guardar los cambios.

16. **Caso de uso**: Editar producto.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe haber completado el caso de uso número 13.
    **Escenario de éxito principal**:
        1. El usuario pulsa el botón “Productos”, situado en el apartado “Mis publicaciones”.
        2. La aplicación le muestra una lista con todos los productos que el usuario ha creado y añadido a la base de datos.
        3. El usuario selecciona el producto que desea editar.
        4. La aplicación le muestra la pantalla con todos los datos del producto.
        5. El usuario edita los campos que desee modificar y pulsa el botón “Aplicar”.
        6. La aplicación modifica el producto en la base de datos y le vuelve a mostrar su lista de productos.

    **Comentarios:** Si el usuario es un administrador, se le mostrará la lista de todos los productos existentes en la base de datos.

17. **Caso de uso**: Borrar producto.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe haber completado el caso de uso número 13.
    **Escenario de éxito principal**:
        1. El usuario pulsa el botón “Productos”, situado en el apartado “Mis publicaciones”.
        2. La aplicación le muestra una lista con todos los productos que el usuario ha creado y añadido a la base de datos.
        3. El usuario selecciona el producto que desea eliminar.
        4. La aplicación le muestra la pantalla con todos los datos del producto.
        5. Pulsa el botón “Eliminar” situado en la barra de acciones superior de *Android*.
        6. La aplicación borra el producto, así como todos sus datos asociados (ofertas, votos, comentarios…) de la base de datos y le vuelve a mostrar su lista de productos.

    **Comentarios:** Si el usuario es un administrador, se le mostrará la lista de todos los productos existentes en la base de datos.

18. **Caso de uso**: Editar oferta.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe haber completado el caso de uso número 13.
    **Escenario de éxito principal**:
        1. El usuario pulsa el botón “Ofertas”, situado en el apartado “Mis publicaciones”.
        2. La aplicación le muestra una lista con todas las ofertas que el usuario ha creado y añadido a la base de datos.
        3. El usuario selecciona la oferta que desea editar.
        4. La aplicación le muestra la pantalla con todos los datos de la oferta.
        5. El usuario edita los campos que desee modificar y pulsa el botón “Aplicar”.
        6. La aplicación modifica la oferta en la base de datos y le vuelve a mostrar su lista de ofertas.

    **Comentarios:** Si el usuario es un administrador, se le mostrará la lista de todas las ofertas existentes en la base de datos.

19. **Caso de uso**: Borrar oferta.
    **Actor principal**: Usuario.
    **Pre-condiciones**: El usuario debe haber completado el caso de uso número 13.
    **Escenario de éxito principal**:
        1. El usuario pulsa el botón “Ofertas”, situado en el apartado “Mis publicaciones”.
        2. La aplicación le muestra una lista con todas las ofertas que el usuario ha creado y añadido a la base de datos.
        3. El usuario selecciona la oferta que desea eliminar.
        4. La aplicación le muestra la pantalla con todos los datos de la oferta.
        5. Pulsa el botón “Eliminar” situado en la barra de acciones superior de *Android*.
        6. La aplicación borra la oferta, así como todos sus datos asociados (votos, comentarios…) de la base de datos y le vuelve a mostrar su lista de ofertas.

    **Comentarios:** Si el usuario es un administrador, se le mostrará la lista de todas las ofertas existentes en la base de datos.

# 4. Diseño e implementación

## 4.1. Base de datos y comunicación con el servidor GAE

Para poder entender adecuadamente el funcionamiento de la base de datos de la aplicación, en los siguientes apartados se explicará una visión resumida de su implementación pero dejando ver claramente los detalles más importantes. En primer lugar, se describirán las entidades que componen la base de datos para, posteriormente, explicar la comunicación de la aplicación con *GAE* (*Google App Engine*) y su almacén de datos (*Datastore*).

### 4.1.1. Diagrama y notación de entidades

Como se ha comentado anteriormente, la base de datos que almacenaremos en el *Datastore* no es de tipo relacional. Realmente, lo único que tenemos que almacenar son las clases que representan a las entidades que necesitamos para la aplicación y *GAE* se encargará del resto. Eso sí, tendremos que definir bien todos los campos y sus "relaciones" para que todo funcione correctamente. La figura 5 muestra las entidades que se han utilizado para almacenar la información de la aplicación.

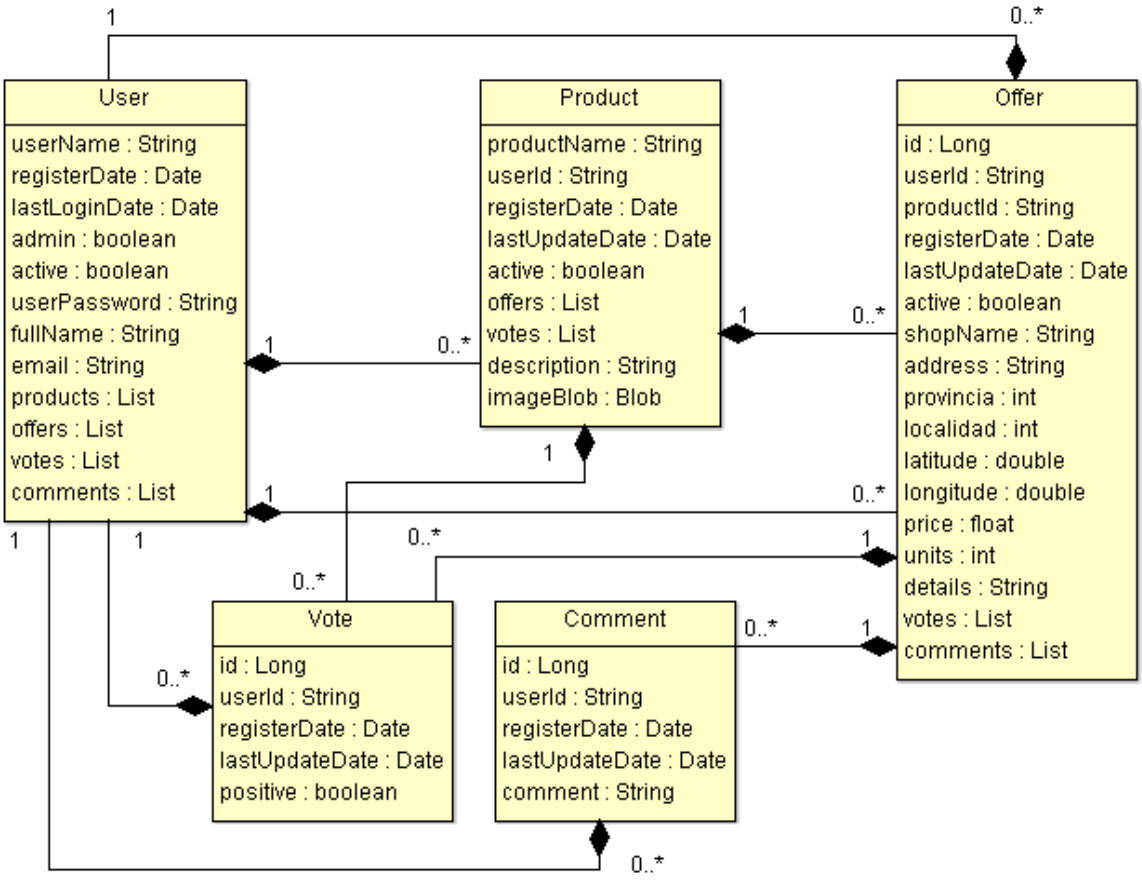


Figura 5. Diagrama de entidades

Cada clase debe tener un identificador único, que puede ser de tipo *Long* o de tipo *String*. Por ejemplo, la clase "User.java" tiene como identificador el campo "userName", que es de tipo *String*, mientras que la clase "Offer.java" tiene como identificador el campo "id", que es de tipo *Long*. Los identificadores de tipo *String* deben ser introducidos por el usuario mientras que los de tipo *Long* son valores autogenerados.

Para ver más claramente cómo se define una clase como una entidad de la base de datos con todos sus campos podemos observar el código de la figura 6. En esta figura se han omitido los *getters* y los *setters* para una mayor claridad del código. Esta entidad ha sido creada utilizando la notación de *Objectify*.

```
@Entity
public class Product {
    @Id
    private String productName;
    @Index
    private String userId;
    @Index
    private Date registerDate;
    @Index
    private Date lastUpdateDate;
    @Index
    private boolean active;
    @Load
    private List<Ref<Offer>> offers = new ArrayList<Ref<Offer>>();
    @Load
    private List<Ref<Vote>> votes = new ArrayList<Ref<Vote>>();
    @Index
    private String description;
    private Blob imageBlob;
```

Figura 6. Definición de entidad con Objectify

Como se observa en la figura 6, hay varios tipos de etiquetas:

- **Etiqueta "@Entity".** Esta etiqueta sirve para indicar que la clase es una entidad de la base datos.
- **Etiqueta "@Id"**. Indica que el campo que viene justo después es el identificador único de la entidad.
- **Etiqueta "@Index".** Indica que el campo que viene justo después es un campo indexado, por lo que podemos realizar consultas de búsqueda en la base de datos para esos campos.
- **Etiqueta "@Load".** Indica que se hace referencia a otra entidad de la base de datos.

Un campo del tipo *List<Ref<Offer>>* indica que es un objeto que representa una lista de referencias a la entidad *Offer* (clase "Offer.java").

### 4.1.2. Descripción de entidades

A continuación se describen las entidades (o clases) que aparecen en la figura 5.

La entidad "User.java" representa a un usuario de la aplicación. Esta entidad almacena los siguientes datos referentes a un usuario:

- **Nombre de usuario**. El campo "userName" de tipo *String*.
- **Fecha de registro**. El campo "registerDate" de tipo *Date*. Aunque internamente *GAE* los almacena como *DateTime*. La conversión de DateTime a Date y viceversa es trivial en *Android*.
- **Fecha de último acceso a la aplicación**. El campo "lastLoginDate" de tipo *Date*.
- **Administrador**. Indica si un usuario es un administrador. El campo "admin" de tipo *boolean*.
- **Active.** Indica si un usuario tiene su cuenta activa. El campo "active" de tipo *boolean*.
- **La contraseña**. "userPassword" de tipo *String*.
- **El nombre completo del usuario**. "fullName" de tipo *String*.
- **El correo electrónico**. "email" de tipo *String*.
- **La lista de productos que ha creado y añadido a la base de datos**. "products", de tipo *List<Ref<Product>>.*
- **La lista de ofertas que ha creado y añadido a la base de datos**. "offers", de tipo *List<Ref<Offer>>.*
- **La lista de votos que ha realizado**. "votes", de tipo *List<Ref<Vote>>.*
- **La lista de comentarios que ha introducido en las ofertas**. "comments", de tipo *List<Ref<Comment>>.*

La clase "Product.java" representa a un producto que ha sido introducido en la aplicación por un usuario. Sus atributos son:

- **Nombre del producto**. El campo "productName" de tipo *String*. Es también el identificador de esta entidad.
- **El usuario que lo creó**. El usuario que creó el producto: "userId", de tipo *String*, que es el identificador de la clase "User.java".
- **Fecha de registro**. El campo "registerDate", de tipo Date.
- **Fecha de última actualización**. El campo "lastUpdateDate", de tipo *Date*.
- **Active**. Indica si el producto está activo. El campo "active", de tipo *boolean*.
- **La lista de ofertas que hay para el producto**. "offers", de tipo *List<Ref<Offer>>.*
- **La lista de votos que ha recibido de los usuarios**. "votes", de tipo *List<Ref<Vote>>.*
- **Descripción del producto**. Campo "description", de tipo *String*.
- **Imagen del producto**. Campo "imageBlob", de tipo *Blob*, que se explicará más adelante en este capítulo.

La clase “Offer.java” representa a una oferta que ha sido introducida en la aplicación por un usuario. Sus atributos son:

- **El identificador único de la oferta**. “id”, de tipo *Long*. Se trata de un valor autogenerado al introducir la entidad en la base de datos.
- **El usuario que la creó**. El usuario que creó la oferta: “userId”, de tipo *String*, que es el identificador de la clase “User.java”.
- **Producto al que pertenece la oferta**. “productId”, de tipo *String*, que es el identificador de la clase “Product.java”.
- **Fecha de registro**. El campo “registerDate” de tipo *Date*.
- **Fecha de última actualización**. El campo “lastUpdateDate” de tipo *Date*.
- **Active**. Indica si la oferta está activa. El campo “active” de tipo *boolean*.
- **Nombre del comercio**. Identifica la tienda que ofrece el producto. El atributo “shopName”, de tipo *String*.
- **Dirección del comercio**. Dirección del comercio donde se localiza la oferta. El atributo “address”, de tipo *String*.
- **Provincia**. Provincia donde se encuentra la oferta. El atributo “provincia”, de tipo *int*, que es el índice que ocupa la provincia en el array de provincias que se ha definido en el código.
- **Localidad**. Localidad donde se encuentra la oferta. El atributo “localidad”, de tipo *int*, que es el índice que ocupa la localidad en el array de localidades que se ha definido en el código.
- **Latitud**. El campo “latitude”, de tipo *double*. Se utiliza para la geolocalización de la oferta.
- **Longitud**. El campo “longitude”, de tipo *double*. Se utiliza para la geolocalización de la oferta.
- **Precio**. El campo “price”, de tipo *float*.
- **Unidades**. El número de unidades del producto que entran en la oferta. El campo “units”, de tipo *int*.
- **Detalles de la oferta**. Los detalles de la oferta que introduce el usuario. Atributo “details”, de tipo *String*.
- **La lista de votos que ha recibido de los usuarios**. “votes”, de tipo *List<Ref<Vote>>.*
- **La lista de comentarios que ha recibido de los usuarios**. “comments”, de tipo *List<Ref<Comment>>.*

La entidad “Vote” (clase “Vote.java”) representa un voto que el usuario ha realizado en una oferta de un producto determinado. Sus atributos son:

- **El identificador único del voto**. “id”, de tipo *Long*. Se trata de un valor autogenerado al introducir la entidad en la base de datos.
- **El usuario que votó**. El campo “userId”, de tipo *String*, que es el identificador de la clase “User.java”.
- **Fecha de registro**. El campo “registerDate” de tipo *Date*.

- **Fecha de última actualización**. El campo “lastUpdateDate” de tipo *Date*.
- **Voto positivo o negativo**. El atributo “positive”, de tipo *boolean*.

La entidad “Comment” (clase “Comment.java”) representa un comentario que ha escrito un usuario en una oferta determinada. Sus campos son:

- **El identificador único del comentario**. “id”, de tipo *Long*. Se trata de un valor autogenerado al introducir la entidad en la base de datos.
- **El usuario que dejó el comentario**. El campo “userId”, de tipo *String*, que es el identificador de la clase “User.java”.
- **Fecha de registro**. El campo “registerDate” de tipo *Date*.
- **Fecha de última actualización**. El campo “lastUpdateDate” de tipo *Date*.
- **El texto del comentario**. El atributo “comment”, de tipo *String*.

### 4.1.3. Comunicación con el servidor GAE: Endpoints

La comunicación desde los clientes, que en este caso son los dispositivos que tengan instalada la aplicación del presente *TFG*, hasta el servidor de *GAE*, se realiza a través de los denominados *Cloud Endpoints*.

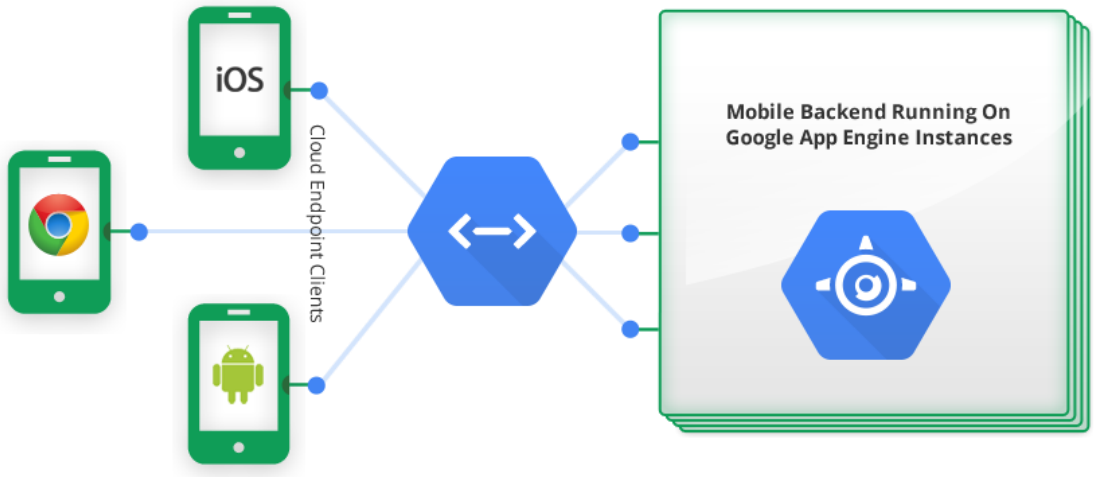


Figura 7. Comunicación Cliente - GAE

En nuestro caso, el código del proyecto cliente *Android (TFGProductos)*, escrito en *Java*, realiza una petición al código del proyecto que incluye los *endpoints (TFGProductos-AppEngine)*, también en *Java*, y que serán los encargados de trasladar dicha petición directamente al servidor *GAE*. La respuesta del servidor se recibe de la misma manera pero en sentido contrario.

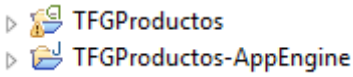


Figura 8. Proyecto Eclipse: Cliente y AppEngine

Utilizando *Eclipse* se puede crear muy fácilmente un proyecto *Android* y su respectivo proyecto de *AppEngine*. Cuando tenemos creado el proyecto *Android* en *Eclipse*, basta con pulsar con el botón derecho del ratón sobre él y, en el apartado "Google", seleccionar la opción "Generate App Engine Backend"… El proyecto *AppEngine* será creado automáticamente.

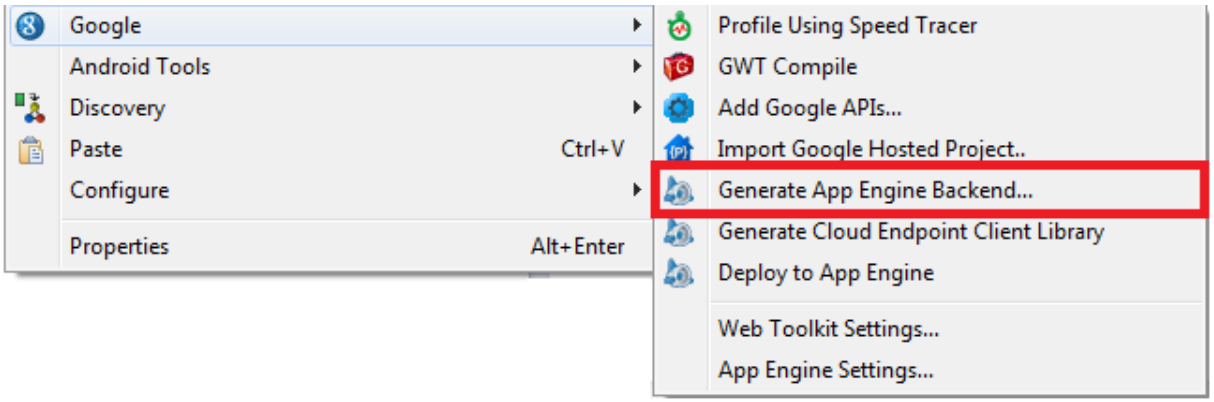


Figura 9. Generación de App Engine Backend en Eclipse

Todo el código referente a las entidades de la base de datos y a la comunicación con el servidor deberá ir incluído en el proyecto *AppEngine*.

### 4.1.4. Manejo

Lo primero que se debe hacer para poder almacenar y utilizar la información referente a una entidad en la base de datos es definir la entidad dentro del proyecto *AppEngine*. La figura 6 mostraba un ejemplo sobre cómo realizar esta definición, aunque hay que añadir los métodos *get* y *set* para poder luego obtener los atributos como se hace con cualquier clase de *Java*.

Una vez que hemos definido la entidad, para que se pueda comunicar con el servidor de GAE, es necesario crear el *endpoint* asociado a esa entidad. *Eclipse* permite generar todo el código *JDO/JPA* para realizar esta comunicación de manera automática. Para ello, se debe pulsar con el botón derecho del ratón sobre la entidad y, dentro del apartado "Google", seleccionar la opción "Generate Cloud Endpoint Class".

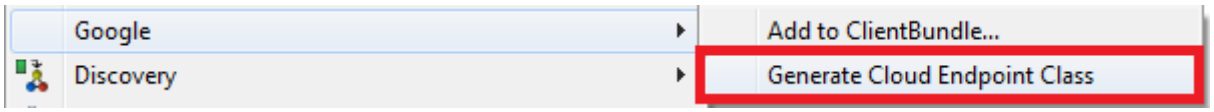


Figura 10. Generación de Cloud Endpoint Class en Eclipse

La figura 11 muestra una parte del código autogenerado del *endpoint* de la clase "Product.java" que muestra la función que obtiene la lista de productos de la base de datos.

```
@SuppressWarnings({ "unchecked", "unused" })
@ApiMethod(name = "listProduct")
public CollectionResponse<Product> listProduct(@Nullable @Named("cursor") String cursorString,
        @Nullable @Named("limit") Integer limit) {

    EntityManager mgr = null;
    Cursor cursor = null;
    List<Product> execute = null;

    try {
        mgr = getEntityManager();
        Query query = mgr.createQuery("select from Product as Product");
        if (cursorString != null && cursorString != "") {
            cursor = Cursor.fromWebSafeString(cursorString);
            query.setHint(JPACursorHelper.CURSOR_HINT, cursor);
        }

        if (limit != null) {
            query.setFirstResult(0);
            query.setMaxResults(limit);
        }

        execute = (List<Product>) query.getResultList();
        cursor = JPACursorHelper.getCursor(execute);
        if (cursor != null)
            cursorString = cursor.toWebSafeString();

        // Tight loop for fetching all entities from datastore and accomodate
        // for lazy fetch.
        for (Product obj : execute)
            ;
    } finally {
        mgr.close();
    }

    return CollectionResponse.<Product> builder().setItems(execute).setNextPageToken(cursorString).build();
}
```

Figura 11. Código para listar productos autogenerado en JDO/JPA

Sin embargo, ese código va a ser muy simplificado ya que en el presente trabajo vamos a utilizar *Objectify* en lugar de *JDO/JPA* para realizar la comunicación con *GAE*. La figura 12 muestra la traducción del código en *JDO/JPA* de la Figura 11 a código *Objectify*.

```
@ApiMethod(name = "listProduct")
public List<Product> listProduct() {
    List<Product> result = new ArrayList<Product>();
    result = ofy().load().type(Product.class).list();

    return result;
}
```

Figura 12. Traducción de JDO/JPA a Objectify del código para listar productos

Como se puede observar, el código es mucho más simple, sólo hay que llamar al método ofy() y decirle que liste todos los productos del tipo que representa la entidad ("Product" en este caso).

Otro paso que hay que dar para que el cliente pueda tener acceso a los *endpoints* del proyecto *AppEngine* es generar dichas librerías en el proyecto cliente. *Eclipse* permite hacerlo de manera automática pulsando el botón derecho del ratón sobre el proyecto *AppEngine* y, dentro del apartado "Google", seleccionar la opción "Generate Cloud Endpoint Client Library".

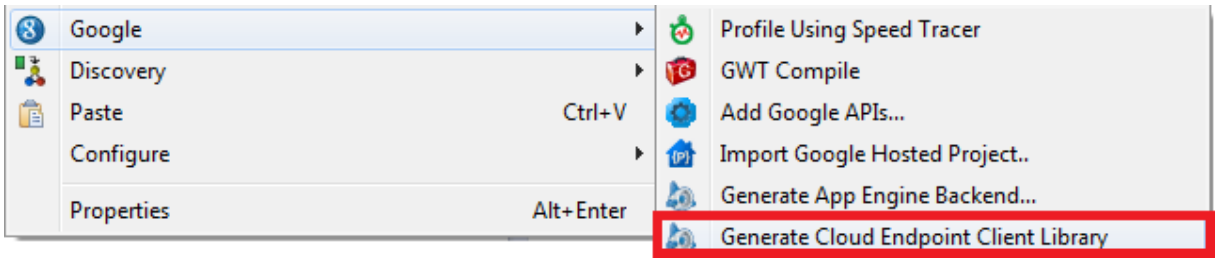


Figura 13. Generar Cloud Endpoint Client Library

El cliente (proyecto *Android*) podrá instanciar la clase "Productendpoint" (del proyecto *AppEngine*) para así poder llamar a sus métodos. Por ejemplo, podrá llamar al método "listProduct", comentado anteriormente, que le proporcionará la lista de todos los productos que hay almacenados en la base de datos remota (de la base de datos al *endpoint* y de éste al cliente).

Cada vez que se desee almacenar una nueva versión de nuestra aplicación en el servidor remoto de *Google App Engine*, bastará con hacer clic con el botón derecho del ratón sobre el proyecto *AppEngine* y, en el apartado "Google", seleccionar la opción "Deploy to App Engine".

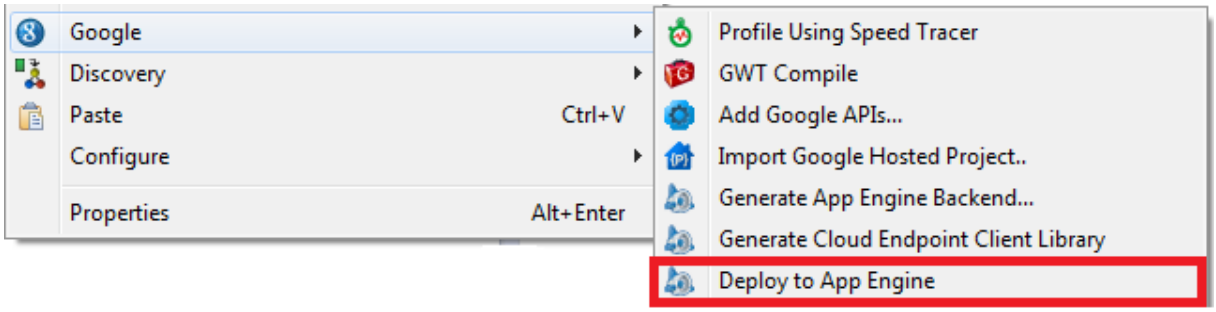


Figura 14. Deploy to App Engine

Por último, comentar que es posible alternar el uso de la base de datos local y remota simplemente modificando el valor de la variable "LOCAL_ANDROID_RUN" perteneciente a la clase "CloudEndpointUtils.java", que se encuentra en el proyecto *Android* (cliente).

```
public class CloudEndpointUtils {

  public static final boolean LOCAL_ANDROID_RUN = true;
  private static final String LOCAL_APP_ENGINE_SERVER_URL = "http://10.0.2.2:8888";

  /**
```

Figura 15. Variable para trabajar en local o remoto con GAE

### 4.1.5. Blob: Almacenamiento de imágenes en GAE

Hay tres alternativas para almacenar ficheros en *Google App Engine*, que en nuestro caso son imágenes:

1. **Blobstore API.** Permite a la aplicación servir objetos de datos, llamados *blobs*, que permiten almacenar objetos de mucho mayor tamaño que el servicio *Datastore*. Los blobs se suben al servidor mediante peticiones *HTTP*. Cuando se sube un blob al servidor, éste devuelve una clave del *blob*, que puede ser utilizada posteriormente para servir ese *blob*. El máximo almacenamiento de una imagen es de 32 Megabytes. No es compatible con el uso de *endpoints*, por lo que esta opción queda descartada para nuestro uso.
2. **Blobstore API con Google Cloud Storage.** Es posible utilizar el *Blobstore API* para almacenar *blobs* en *Google Cloud Storage*. Este servicio permite almacenar ficheros de gran tamaño y, adicionalmente, ofrece el uso de listas de control de acceso (*ACLs*), la posibilidad de resumir una subida si fue interrumpida por cualquier motivo y muchas otras funciones. Es de pago, por lo que queda descartada para este trabajo.
3. **Almacenamiento en el Datastore.** Es posible almacenar la imagen directamente en el propio *Datastore*, aunque el máximo tamaño del objeto está limitado a 1 Megabyte.

Tras estudiar las tres opciones posibles para el almacenamiento de las imágenes, nos quedamos con la opción de almacenarlo directamente en el *Datastore*. El *Datastore* también usa un objeto de tipo *Blob* para el almacenamiento de las imágenes. Para generar un *Blob*, esta clase utiliza un objeto de tipo *byte[]*, que hay que pasarle como parámetro, y que va a contener la imagen a almacenar.

Cuando el cliente (proyecto *Android*) quiere obtener una imagen almacenada en una entidad, lo que realmente recibe es un *String* que contiene la codificación de la imagen. Pero, ¿cómo obtener la imagen a partir de este *String*?. Este *String* es una representación de datos binarios en *Base64*. La utilidades de la clase *Base64* de *Android* permiten decodificar este *String* en *Base64* directamente a un *byte[]*, que podemos utilizar para obtener un *Bitmap* de él, que será ya nuestra imagen. Para la obtención de ese *Bitmap*, podemos utilizar la clase *BitmapFactory* de *Android*. La figura 16 muestra un método implementado durante el desarrollo del trabajo para realizar todo este proceso.

```
public class ImageUtils {
    public static Bitmap decodeBase64(String input) {
        byte[] decodedByte = Base64.decode(input, 0);
        return BitmapFactory.decodeByteArray(decodedByte, 0, decodedByte.length);
    }
```

Figura 16. Decodificación de String en Base64

## 4.2. Aplicación Android

### 4.2.1. Ciclo de vida: La clase Activity

La figura 17 muestra el ciclo de vida de una instancia de la clase *Activity* de *Android*.

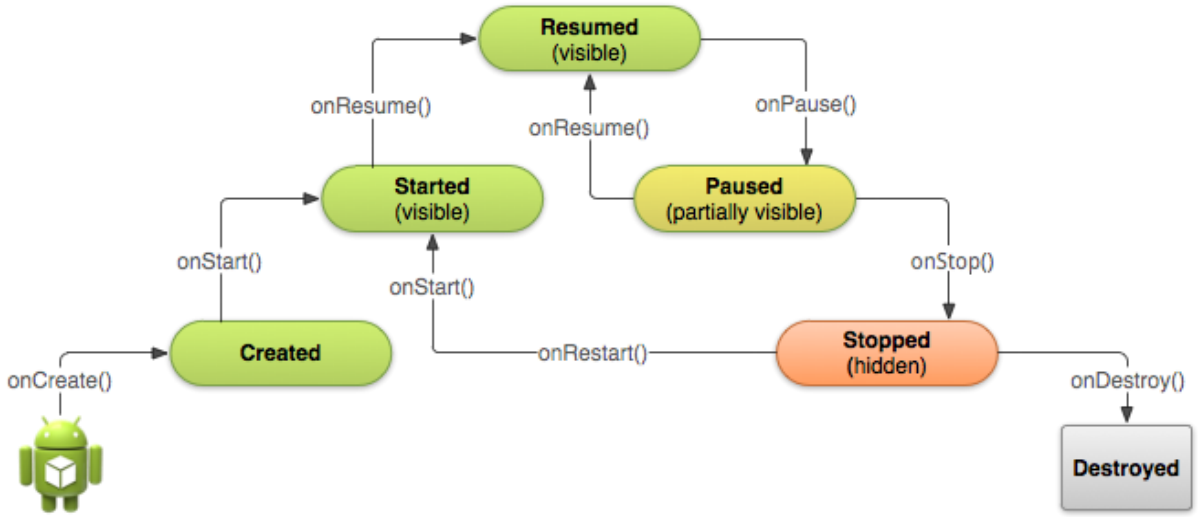


Figura 17. Ciclo de vida de una actividad Android

Dependiendo de las necesidades de la aplicación que se esté desarrollando, se utilizarán unos métodos u otros del ciclo de vida. El objetivo del ciclo de vida es asegurar el buen comportamiento de la aplicación en varios sentidos:

- No se produce un fallo si el usuario recibe una llamada o cambia a otra aplicación.
- No consume recursos cuando no se están usando.
- No perder el progreso del usuario en la aplicación si el usuario se va de la aplicación y luego vuelve.
- No fallar o perder el progreso del usuario si la pantalla cambia de modo horizontal a vertical. Cuando este cambio de orientación ocurre, la actividad es creada desde el principio.

En el estado “Resumed” la actividad está mostrándose por pantalla y el usuario puede interactuar con ella. Cuando se está en el estado “Paused”, la actividad está siendo parcialmente ocultada por otra actividad y la actividad ahora al frente es semi-transparente o no cubre por completo la pantalla. En el estado “Stopped” la actividad está completamente oculta. El resto de estados son transitorios y se pasa de unos a otros rápidamente.

Algo que puede ser considerado como una desventaja es el hecho de que, cada vez que una actividad se oculta por completo para dar paso a otra actividad, se pierden todos los valores de las variables almacenados en ella. Por ejemplo, al cambiar de una actividad a otra, automáticamente la primera se coloca en la pila de actividades para permitir el regreso a ella por parte del usuario, normalmente pulsando el botón “Atrás”, sin embargo, toda la información guardada en las variables de esa actividad se habrán perdido. Lo mismo ocurre al cambiar la orientación de la pantalla del dispositivo, la actividad que estaba siendo mostrada antes de girarla es destruida para inmediatamente ser creada nuevamente, perdiendo todos los valores de las variables.

### 4.2.2. Persistencia de datos

Como se ha comentado, el valor de las variables de una actividad se pierde cuando ésta vuelve a ser puesta en primer plano tras haber sido parada. Una forma de guardar la información de estas variables es mediante el uso de los métodos “onSaveInstanceState()” y “onRestoreInstanceState()”.

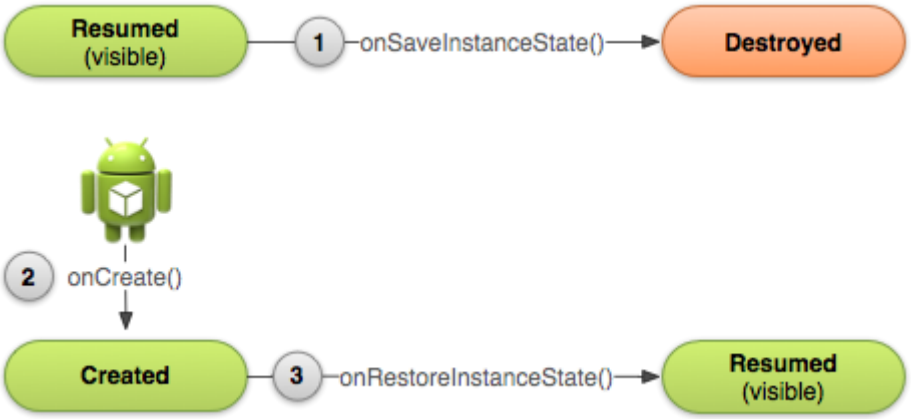


Figura 18. Diagrama de recuperación de estado

Estos métodos incluyen un parámetro, de tipo *Bundle*, que es el encargado de almacenar datos sobre el estado de la aplicación. El método "onSaveInstanceState()" nos permite almacenar la información que necesitemos justo antes de terminar la actividad o pasar a otra actividad. Cuando la actividad vuelva a ser creada se ejecutará automáticamente el método "onRestoreInstanceState()", en el que podremos recuperar el estado de dicha actividad.

Este método que se acaba de comentar no sirve para dar persistencia entre sucesivas ejecuciones de la aplicación. Para dar persistencia a los datos, por ejemplo, para recordar los datos de acceso de un usuario para no tener que introducirlos cada vez que se ejecuta la aplicación o para recordar algunas preferencias como, por ejemplo, su ubicación, se ha utilizado la clase *SharedPreferences* que ofrece *Android*.

```
public static int getLocalidad(Context context) {
    SharedPreferences sharedPref = context.getSharedPreferences(SHARED_PREFERENCES_FILE_KEY, Context.MODE_PRIVATE);
    int res = sharedPref.getInt(SHARED_PREFERENCES_LOCALIDAD_KEY, SHARED_PREFERENCES_DEFAULT_LOCALIDAD_VALUE);
    return res;
}
```

Figura 19. Recuperación de valores de variables: SharedPreferences

Como se puede observar en la figura 19, en primer lugar se abre el fichero de preferencias y se obtiene un valor de tipo *int* mediante el método "getInt" de la clase *SharedPreferences*. A este método se le ha indicado que se quiere obtener el entero almacenado con la clave "SHARED_PREFERENCES_LOCALIDAD_KEY" y, en caso de no haber sido almacenado todavía dicho valor, devolverá un valor por defecto que ha debido ser previamente almacenado en la variable "SHARED_PREFERENCES_DEFAULT_LOCALIDAD_VALUE". Ambas variables deben ser creadas por el usuario con el valor deseado.

```
public static void setLocationPreferences(Context context, int provincia, int localidad) {
    SharedPreferences sharedPref = context.getSharedPreferences(SHARED_PREFERENCES_FILE_KEY, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPref.edit();
    editor.putInt(SHARED_PREFERENCES_PROVINCIA_KEY, provincia);
    editor.putInt(SHARED_PREFERENCES_LOCALIDAD_KEY, localidad);
    editor.commit();
}
```

Figura 20. Almacenamiento de valores de variables: SharedPreferences

En la figura 20 se puede ver un ejemplo en el que se almacena la variable localidad, utilizando para ello el método "putInt", de la clase estática *Editor*, con la clave "SHARED_PREFERENCES_LOCALIDAD_KEY", utilizando la misma clave de archivo "SHARED_PREFERENCES_FILES_KEY".

### 4.2.3. La clase Fragment

Un *Fragment* representa el comportamiento de una porción de la interfaz gráfica de un *Activity*. Se pueden combinar múltiples fragmentos en una misma actividad y reutilizar esos fragmentos en otras actividades. Un fragmento tiene un ciclo de vida similar al de una actividad y se ve directamente afectado por el ciclo de vida de una actividad, por ejemplo, si una actividad se para, también lo harán todos sus fragmentos.

Un fragmento también puede ser utilizado para mejorar la experiencia del usuario dependiendo del tipo de dispositivo que utilice. Por ejemplo, si la aplicación detecta que la pantalla puede albergar el tamaño de dos fragmentos que están situados uno al lado del otro, los mostrará tal cual, pero si detecta que no caben en la pantalla, utilizará un *layout* alternativo en el que los fragmentos se mostrarán de uno en uno.

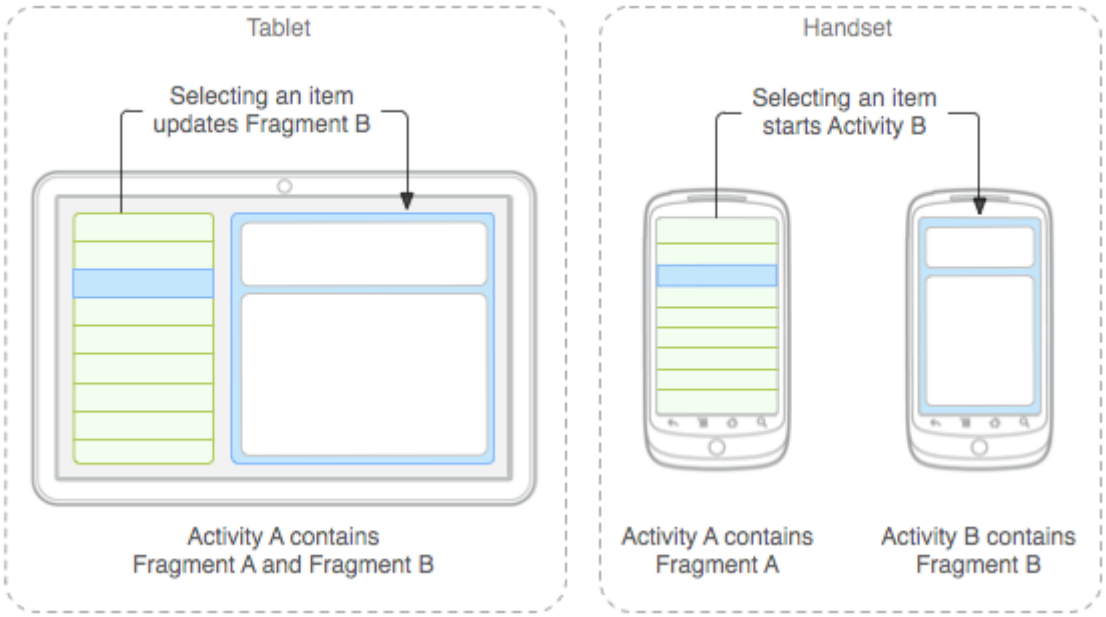


Figura 21. Fragmentos

### 4.2.4. Diseño de la interfaz gráfica: Layouts

Cuando una actividad ejecuta su método "onCreate", una de las cosas que debe hacer es buscar la información acerca de los componentes que se van a mostrar en la pantalla. Para ello, la actividad busca en la carpeta "res/layout" del proyecto *Android*, el fichero .xml que le hayamos indicado que debe utilizar para cargar el *layout* que contiene la información sobre la disposición de los elementos en la pantalla. Este fichero estará compuesto por componentes de tipo *ViewGroup* y *View*. Un *ViewGroup* establece la disposición en la pantalla de los elementos *View* que va a contener. Por ejemplo, se puede crear un *LinearLayout*, en el que sus elementos se irán situando conforme a la orientación que le indiquemos, vertical u horizontal, mientras que en un *RelativeLayout*, se colocarán los elementos de manera relativa a otros elementos ya existentes. Existen también otros tipos de *layouts* más complejos como, por ejemplo, para la manipulación de listas de elementos, que pueden ser utilizados conforme a las necesidades que tenga la aplicación. Un *View* es un elemento de la interfaz gráfica como puede ser una etiqueta, un campo de texto editable o un botón y que siempre irá dentro de un elemento de tipo *ViewGroup*.

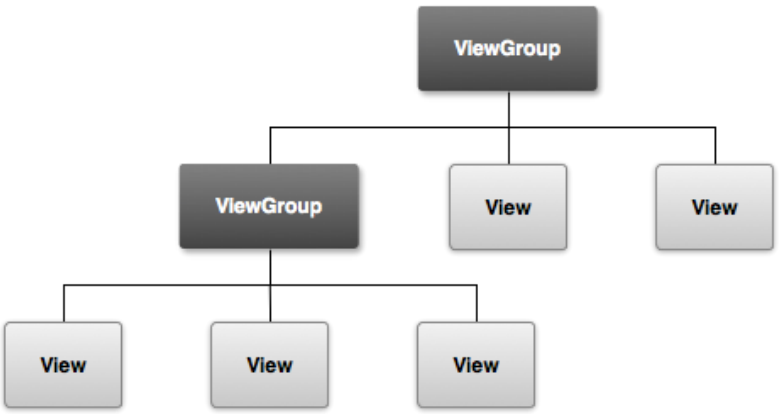


Figura 22. Jerarquía de componentes de un Layout

Además de cargar los componentes gráficos del *layout* desde este fichero, es posible hacerlo también desde el propio código, por ejemplo, añadiendo elementos View a un ViewGroup, cambiando el texto de una etiqueta, etc. Para identificar desde el código los elementos del fichero .xml es conveniente ponerles un identificador a cada *ViewGroup* o *View* a los que se vaya a hacer referencia desde el código de la aplicación.

### 4.2.5. La barra de acciones: Action Bar

Otro elemento importante que nos ofrece *Android* y que hemos utilizado en el desarrollo de la aplicación es la denominada *Action Bar*.

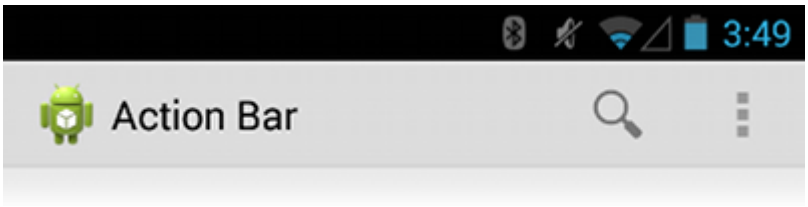


Figura 23. Action Bar de Android

Esta barra de acciones está situada en la parte superior de la pantalla. Todas las aplicaciones *Android* la usan de la misma forma y tiene un aspecto similar, dando así consistencia con el resto de las aplicaciones *Android*.

Para definir qué elementos va a contener la barra de acciones debemos establecer su *layout*, tal como se hace con cualquier actividad o fragmento, pero guardando el fichero .xml dentro de la carpeta "res/menú" del proyecto *Android*. El *layout* del menú se le indicará a la aplicación en el método "onCreateOptionsMenu".

Cuando el usuario pulsa uno de los botones de esta barra de acciones, automáticamente se ejecuta el método "onOptionsItemSelected" de la actividad que se tiene en pantalla. En este método se puede comprobar qué botón ha sido pulsado para así ejecutar las acciones pertinentes.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.action_new:
        actionBarNewButtonPressed();
        return true;
    case R.id.action_search:
        onSearchRequested();
        return true;
    case R.id.action_user_profile:
        showUserProfile();
        return true;
```

Figura 24. Código para la detección de eventos de Action Bar

### 4.2.6. Estructura de la aplicación

Una vez que se han comentado los aspectos más relevantes sobre el funcionamiento de los componentes más importantes de la aplicación desarrollada, su estructura será mucho más fácil de entender. En este apartado se comentará resumidamente el diseño y la implementación de las actividades y fragmentos que componen la aplicación.

La aplicación está compuesta por dos actividades principales, que irán alternando sucesivos fragmentos dentro de ellas conforme el usuario vaya interactuando con la pantalla del dispositivo.

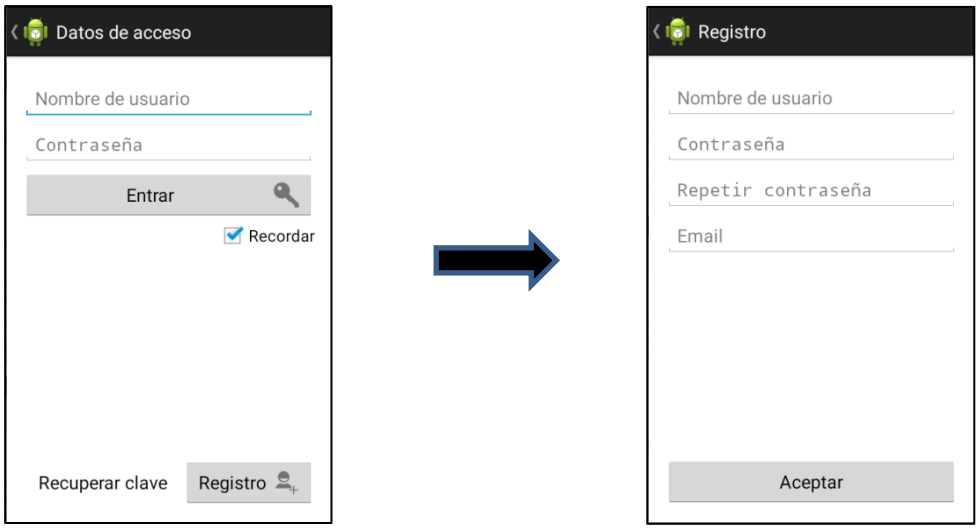


Figura 25. Reemplazo de fragmento en una actividad

En la Figura 25, la actividad principal de la aplicación, "MainActivity.java", situada a la izquierda en la imagen, ha comenzado su ejecución y se encuentra en primer plano. Durante su arranque, hemos puesto el código necesario para llenar toda la pantalla con el fragmento que muestra los elementos necesarios para que el usuario introduzca sus datos de acceso, se registre, etc.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    this.context = this;
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    if (savedInstanceState == null) {
        getFragmentManager().addOnBackStackChangedListener(getListener());
        fragmentUserLogin = new UserLoginFragment();
        FragmentTransaction transaction = getFragmentManager().beginTransaction();
        transaction.add(R.id.containerMainActivity, fragmentUserLogin);
        setBackButtonTitle(getString(R.string.access_data));
        transaction.commit();
    }
}
```

Figura 26. Código para añadir un fragmento a una actividad

La figura 26 muestra el código utilizado para mostrar el fragmento durante el arranque de la actividad, en su método "onCreate". Lo primero que se hace es establecer el *layout* de la actividad, "activity_main", que es un *layout* que únicamente contiene un *ViewGroup* que ocupa toda la pantalla y que tiene como identificador "containerMainActivity". Por eso, el código anterior lo que hace es decirle al manejador de fragmentos de la actividad, que añada el fragmento

"fragmentUserLogin" al *layout* que tiene como identificador "containerMainActivity", mostrando así la pantalla de datos de acceso del usuario que se encuentra definida en ese fragmento.

En el momento en el que el usuario pulse el botón "Registro", la actividad reemplazará el fragmento que muestra los datos de acceso por el fragmento que muestra los datos que permiten el registro de un nuevo usuario.

```
public void buttonUserRegisterPressed(View view) {
    fragmentUserRegister = new UserRegisterFragment();
    FragmentTransaction transaction = getFragmentManager().beginTransaction();
    transaction.replace(R.id.containerMainActivity, fragmentUserRegister);
    transaction.addToBackStack(null);

    transaction.commit();
}
```

Figura 27. Código de reemplazo de un fragmento

El aspecto del fragmento que muestra los datos de registro puede verse a la derecha de la imagen en la figura 25. Lo que hace el código de la figura 27 es decirle al manejador de fragmentos de la actividad que reemplace todo el contenido del layout "containerMainActivity", que se encuentra ocupado por el fragmento "fragmentUserLogin", por el fragmento "fragmentUserRegister". Además, introduce el fragmento reemplazado en la pila de fragmentos mediante el método "addToBackStack" para poder ser recuperado posteriormente. Esta transición es la mostrada en la figura 25.

Volviendo a la pantalla de "Datos de acceso", cuando el usuario introduce sus datos y pulsa el botón "Entrar", la aplicación comprueba que sus datos sean correctos consultando la base de datos y, si lo son, la actividad "MainActivity" invoca a la actividad "ProductosMainActivity", que pasará a tomar el control de la pantalla. "MainActivity" pasa automáticamente a la pila de actividades para volver a ella cuando el usuario quiera volver atrás.

La figura 28 muestra la actividad "ProductosMainActivity", la cual ha sido dividida en dos partes (*layouts*), la barra de título (parte superior, en naranja) y el resto de la pantalla. En cada uno de ellos se irán reemplazando fragmentos conforme el usuario interactúe con la aplicación. La barra de título se utilizará solo para dar información al usuario acerca del fragmento que se esté mostrando en ese momento, mientras que la parte central será donde el usuario realice las acciones sobre la aplicación y también mostrará el contenido de la base de datos.

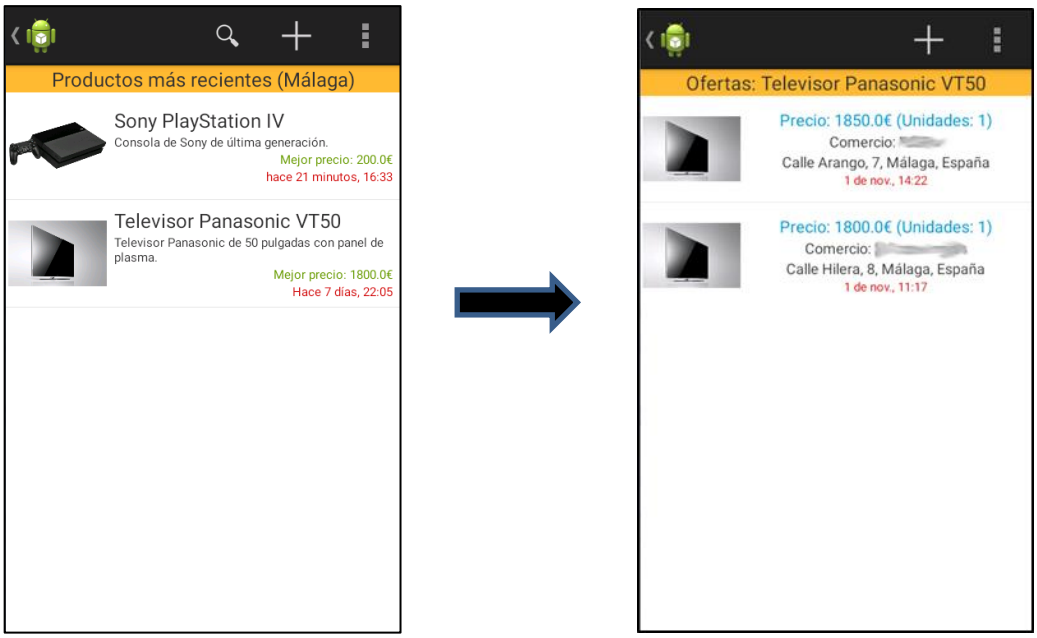


Figura 28. Manejo de fragmentos

La actividad ha sido preparada para actuar conforme vaya detectando cambios de fragmentos en el *layout* central, que se van a producir conforme a las acciones del usuario. Cada vez que detecte que se ha cambiado el fragmento central, se identificará qué fragmento es el que está siendo mostrado en pantalla y se actualizará inmediatamente su información. La siguiente línea de código permite asignar un *listener* para la pila de fragmentos, de manera que se ejecutará su código cada vez que se produzca un cambio en la pila, sea para introducir un nuevo fragmento o bien para quitarlo de ella.

La siguiente línea ha sido añadida al método "onCreate" de la actividad "ProductosMainActivity" para que el *listener* sea asociado a la pila de fragmentos en cada creación de la actividad.

```
getFragmentManager().addOnBackStackChangedListener(getListener());
```

El método "getListener" es el que implementa dicho *listener* y se encarga de comprobar qué fragmento está siendo visualizado cuando se ha producido un cambio en la cola de fragmentos para así realizar las acciones pertinentes. La figura 29 muestra parte de su código.

```
private OnBackStackChangedListener getListener() {
    OnBackStackChangedListener result = new OnBackStackChangedListener() {
        public void onBackStackChanged() {
            if ((fragmentComments != null) && fragmentComments.isVisible()) {
                refreshUserComments();
            } else if ((fragmentNewProduct != null) && fragmentNewProduct.isVisible()) {
                // Producto nuevo
                refreshNewProduct();
            } else if ((fragmentEditProduct != null) && fragmentEditProduct.isVisible()) {
                // Edición de producto existente
                refreshEditProduct();
```

Figura 29. Código que se ejecuta al modificar la pila de fragmentos

Por lo tanto, un usuario realiza una acción sobre la pantalla, como por ejemplo pulsar un botón, el programa realiza las acciones asociadas a ese evento de botón y, si necesita modificar el fragmento central, añadirá el fragmento que se estaba mostrando en ese momento en el *layout* central de la actividad a la pila de fragmentos para ser sustituido por uno nuevo, pasando este último a ser visible. En este momento se ejecuta el *listener* y comprueba qué fragmento está visible, actualizando así su información.

Se ha implementado esta forma de actualizar el fragmento o fragmentos visibles para poder tener todo encapsulado en un mismo sitio, tanto cuando se añaden fragmentos como cuando el usuario vuelve a un fragmento anterior. En el caso de volver a un fragmento anterior no podríamos saber qué fragmento es el que va a ser visible cuando la actividad, automáticamente, saque el fragmento de la pila para volver a mostrarlo. La principal ventaja de detectar los cambios en la pila de fragmentos es el hecho de que se ejecutará tanto cuando se añade un fragmento a la pila como cuando es sacado de ella.

Un ejemplo de esto ocurre en el caso en que el usuario selecciona uno de los dos productos que se muestran en la figura 28, el fragmento de la lista de productos será introducido en la pila de fragmentos y el fragmento que muestra las ofertas que existen de ese producto será mostrado por pantalla mostrando con toda la información actualizada porque, al introducir el fragmento anterior en la pila, se habrá ejecutado el *listener* que habrá comprobado que el nuevo fragmento visible es el de la lista de ofertas.

Con este ejemplo de funcionamiento de la actividad “ProductosMainActivity” ya se puede entender el funcionamiento de las transiciones de la interfaz de usuario para el resto de situaciones, ya que se estará continuamente realizando la tarea de sustituir el *layout* central de la actividad “ProductosMainActivity” por el fragmento oportuno y actualizando el fragmento del título dependiendo de las acciones del usuario.

### 4.2.7. Barra de búsqueda (SearchView)

La barra de búsqueda permite al usuario escribir el nombre del producto que está buscando para que se le muestre la lista de productos que coinciden o se parecen al texto introducido. Además se ha añadido funcionalidad extra para ir mostrando una lista de productos coincidentes con el texto que está introduciendo para facilitar la búsqueda (lista de sugerencias). Si alguno de los productos mostrados en las sugerencias coincide con el que el usuario está buscando, sólo tiene que pulsarlo para ir a su lista de ofertas.


Figura 30. SearchView con sugerencias

Para realizar su implementación, una de las cosas que hay que hacer es crear la configuración de búsqueda. Para ello, hay que crear un fichero en el directorio "res/xml/" del proyecto *Android*, que en este caso se denomina "search_product.xml", incluyendo el siguiente código.

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_name"
    android:hint="@string/search_hint"
    android:searchSuggestIntentAction = "android.intent.action.VIEW">
</searchable>
```

En este fichero se establece la etiqueta del elemento <searchable>, el texto de ayuda que se mostrará cuando no haya nada escrito en el cuadro de texto de búsqueda y la acción del *Intent* para las sugerencias. Este elemento acepta otros tipos de atributos que no son necesarios para esta aplicación. En realidad, el único atributo obligatorio es la etiqueta.

También hay que establecer una actividad que sea de tipo *Searchable*, que se encargue de hacer las búsquedas basadas en el texto que introduzca el usuario. En nuestro caso, la actividad "ProductosMainActivity" será la encargada de realizar esta tarea.

Para establecer “ProductosMainActivity” como la actividad de búsqueda, es necesario modificar el fichero “AndroidManifest.xml”. Este fichero contiene todas las configuraciones generales de la aplicación. La primera modificación es añadir el siguiente código dentro del elemento <application>:

```
<meta-data
    android:name="android.app.default_searchable"
    android:value=".ProductosMainActivity" />
```

Con esto, se le está diciendo a la aplicación, que la actividad de búsqueda por defecto es “ProductosMainActivity”.

Además, hay que añadir las siguientes líneas en el campo <activity> de “ProductosMainActivity”:

```
<intent-filter>
    <action android:name="android.intent.action.SEARCH" />
</intent-filter>

<meta-data
    android:name="android.app.default_searchable"
    android:value="com.appspot.tfgproductostest.ProductosMainActivity" />
<meta-data
    android:name="android.app.searchable"
    android:resource="@xml/search_product" />
```

Cuando el usuario ejecuta una búsqueda, el sistema envía un *Intent* con la acción “ACTION_SEARCH” a la actividad *Searchable*, que pasa a estar en primer plano en caso de que no lo estuviera. Para poder recibir dicho *Intent* con la información del texto introducido por el usuario, se ha añadido el siguiente código a la clase “ProductosMainActivity.java”. Este método es ejecutado automáticamente cada vez que se recibe un nuevo *Intent* en la actividad.

```
@Override
protected void onNewIntent(Intent intent) {
    setIntent(intent);
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
        String query = intent.getStringExtra(SearchManager.QUERY);
        submitUserSearch(query);
    }
}
```

Lo primero que hace el código es comprobar si el nuevo *Intent* es de tipo “ACTION_SEARCH” para así realizar las acciones pertinentes que, en este caso, son obtener el texto introducido por el usuario y realizar las acciones de búsqueda de productos.

Hasta aquí tenemos funcionando el sistema de búsqueda pero aún falta añadir el sistema de sugerencias. Para ello ya habíamos añadido a la configuración de búsqueda (fichero search_product.xml) la línea que establece el tipo de *Intent* para las sugerencias. Además, cuando se crea el menú de la barra de acciones hay que ejecutar el siguiente código:

```
searchMenuItem = menu.findItem(R.id.action_search);
SearchManager manager = (SearchManager) getSystemService(Context.SEARCH_SERVICE);
SearchView search = (SearchView) menu.findItem(R.id.action_search).getActionView();
search.setSearchableInfo(manager.getSearchableInfo(getComponentName()));
search.setOnQueryTextListener(new OnQueryTextListener() {

    @Override
    public boolean onQueryTextChange(String query) {
        loadData(query);
        return true;
    }

    @Override
    public boolean onQueryTextSubmit(String query) {
        searchMenuItem.collapseActionView();
        return false;
    }

});
```

Básicamente, lo que hace es establecer un *listener* que estará escuchando dos tipos de eventos: los cambios en el texto de búsqueda y cuando un usuario confirma la realización de la búsqueda. Conforme el usuario vaya introduciendo cambios en el texto de búsqueda, se estará llamando a la función "loadData". Esta función se encargará de mostrar la lista de sugerencias (ver figura 30) de acuerdo al texto introducido hasta el momento, para ello generará la lista teniendo en cuenta todos los productos de la base de datos y establecerá el adaptador de sugerencias para la barra de búsqueda (*SearchView*), que indica qué elementos va a contener y cómo se van a disponer en la pantalla para cada fila de la lista. Para ello, el adaptador utiliza un fichero de *layout* como los explicados anteriormente.

```
SearchManager manager = (SearchManager) getSystemService(Context.SEARCH_SERVICE);
final SearchView search = (SearchView) menu.findItem(R.id.action_search).getActionView();
search.setSearchableInfo(manager.getSearchableInfo(getComponentName()));
search.setSuggestionsAdapter(new ProductSuggestionsAdapter(this, cursor, suggestionList));
```

Con esto ya tenemos una idea de cómo funciona todo el sistema de búsqueda y sugerencias de la aplicación.

### 4.2.8. Selección de imágenes: Cámara y galería

Para añadir imágenes al añadir o editar un producto, la aplicación ofrece al usuario la posibilidad de hacerlo desde la propia galería multimedia del dispositivo o bien haciendo directamente una foto al producto utilizando la cámara del dispositivo.

Para mostrar la galería de imágenes del dispositivo al usuario hay que tener en cuenta que, dependiendo de la *API* (o versión) de *Android* que éste utilice, habrá que hacerlo de una forma u otra.

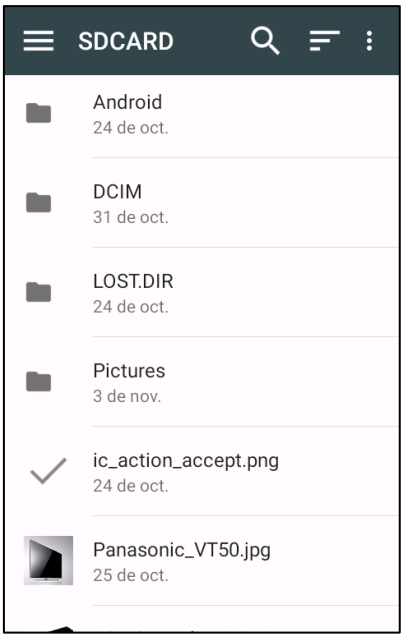


Figura 31. Galería de imágenes

Por tanto, lo primero que hace la aplicación antes de mostrar la galería de imágenes es comprobar la *API* que utiliza el dispositivo y así asegurar que la forma de mostrarla es compatible con él.

```
if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.KITKAT) {
    Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    intent.setType("image/*");
    startActivityForResult(intent, REQUEST_CODE_PICK_IMAGE);
} else {
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(Intent.createChooser(intent, getString(R.string.select_image)), REQUEST_CODE_PICK_IMAGE);
}
```

Para acceder a la cámara hay que iniciar también una nueva actividad con un *Intent* de tipo “ACTION_IMAGE_CAPTURE”. El usuario podrá realizar la foto en ese momento y, al volver, la aplicación habrá añadido la foto a los detalles del producto que se está añadiendo o editando. Por defecto, al utilizar el procedimiento por defecto de la cámara, la imagen guardada es muy pequeña, ya que se obtiene únicamente una pequeña previsualización de la foto que ha realizado el usuario. Para poder obtener la foto en su tamaño original es necesario especificarle en el *Intent* una ruta local del dispositivo para que almacene ahí la foto completa temporalmente. Cuando se vuelva de la aplicación, se accede al fichero de la foto y se añade a la aplicación. Las imágenes que se almacenan en la base de datos tienen un tamaño máximo de 800x600, aunque podríamos haber utilizado otra resolución, hay que tener cuidado de no exceder el tamaño máximo de 1 Megabyte establecido por el *Datastore*.

Por último, para que todo esto funcione, hay que dar permiso a nuestra aplicación para poder acceder a la galería del usuario y a la cámara modificando el fichero AndroidManifest.xml.

### 4.2.9. Acceso a Google Maps

La figura 32 muestra el aspecto de la aplicación cuando el usuario solicita ver en el mapa la dirección en la que se encuentra una oferta determinada. Como se puede observar, aparece un marcador justo en el punto que marca la longitud y latitud almacenada por el usuario que creó la oferta.

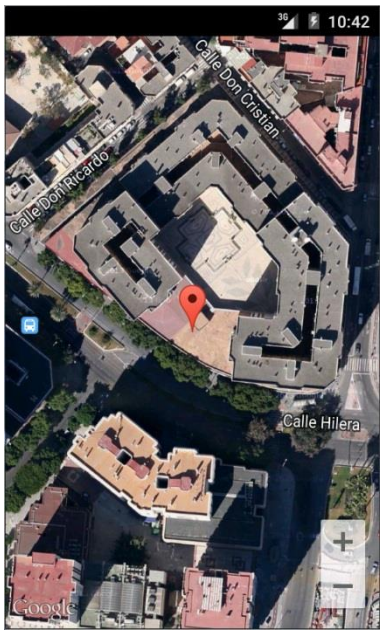


Figura 32. Google Maps en la aplicación

Para poder acceder a los servidores de *Google Maps* con este *API*, es necesario añadir una clave para los mapas *API* a la aplicación. Esta clave tiene la ventaja de ser gratis y poder ser utilizada por un número ilimitado de usuarios. Esta clave puede obtenerse desde la consola de administración de *Google*. Los pasos a seguir para obtenerla son los siguientes:

1. Obtener la información sobre el certificado de nuestra aplicación, que puede ser obtenido utilizando el *Android SDK Tools de Eclipse*. Con esto se obtiene una huella *SHA-1* que servirá de certificado. Este certificado hay que añadirlo a la consola de desarrolladores de *Google* en el apartado *Credentials* (ver figura 33).

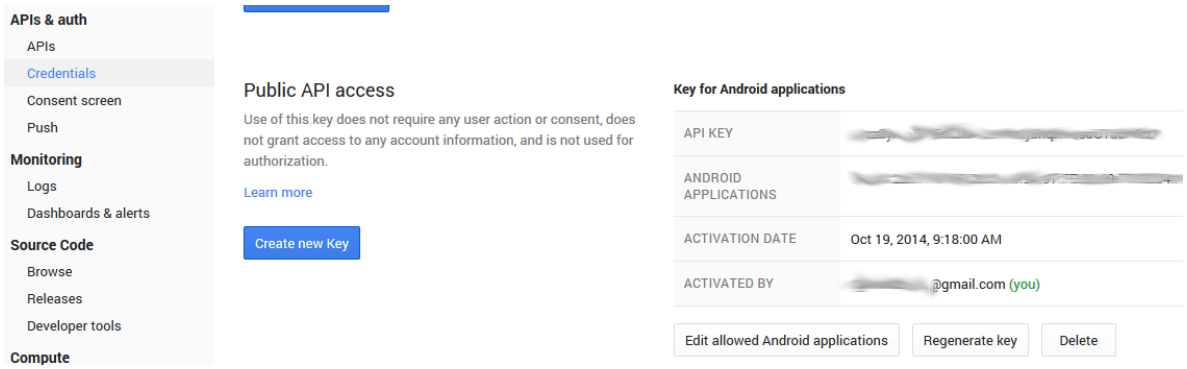


Figura 33. Consola de Google: Credentials

2. Registrar el proyecto en la consola de *Google* y añadir los mapas *API* como servicio al proyecto. En el apartado *APIs*, seleccionar *Google Maps Android API v2* para que aparezca en estado ON.

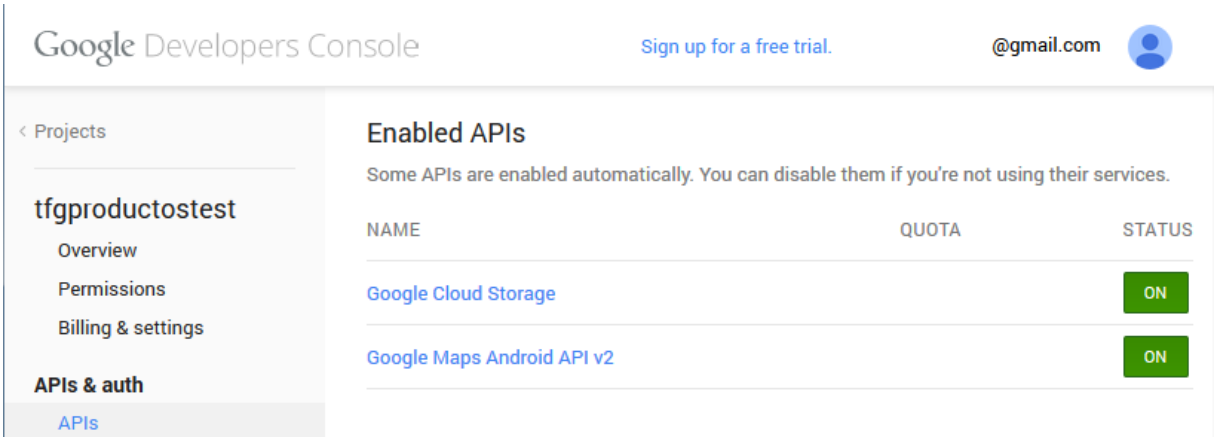


Figura 34. Consola de Google: APIs habilitadas

3. Añadir la clave generada al fichero "AndroidManifest.xml" del proyecto *Android*.

Ya es posible utilizar la *API* de *Google Maps* en la aplicación, pero antes hay que configurarla bien dentro de la misma. Para hacerlo, hay que añadir las siguientes líneas al apartado <application> del fichero "AndroidManifest.xml".

```
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

Este código se encarga de incluir la versión de los servicios de *Google Play* con la que fue compilada la aplicación.

También es necesario añadir a la aplicación *Android*, en el fichero "AndroidManifest.xml", los siguientes permisos:

- **android.permission.INTERNET**. Usado por la *API* para descargar los mapas de los servidores.
- **android.permission.ACCESS_NETWORK_STATE**. Permite a la *API* comprobar el estado de la conexión para determinar si puede descargar datos del servidor.
- **android.permission.WRITE_EXTERNAL_STORAGE**. Permite a la *API* utilizar el almacenamiento externo como caché para los mapas.

Para el caso de la presente aplicación, es necesario añadir otros permisos para realizar tareas extra, además, *Google* recomienda su activación:

- **android.permission.ACCESS_COARSE_LOCATION**. Permite a la *API* utilizar el *WiFi* o los datos del móvil (o ambos) para determinar la localización del móvil.
- **android.permission.ACCESS_FINE_LOCATION**. Permite a la *API* utilizar el *GPS*.

Además, hay que tener instalada la versión 2 de *OpenGL ES* para poder renderizar el mapa. Esto se especifica en la aplicación añadiendo el siguiente código en el fichero "AndroidManifest.xml".

```
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />
```

Finalmente, se puede utilizar la clase *MapFragment,* que facilita las librerías de la *API* de *Google Maps Android* para realizar las operaciones necesarias sobre los mapas. La acción típica que realiza nuestra aplicación es buscar en el mapa la longitud y latitud que le pasemos como parámetros, añadir un marcador en ese punto y hacer zoom para enfocar la zona y sea más cómodo de localizar por parte del usuario.

### 4.2.10. Envío de emails

Para la recuperación de datos de acceso del usuario es necesario implementar el sistema de envío de emails en la aplicación. Para ello se ha utilizado la *API* de *JavaMail*, ya que es muy simple de utilizar, funciona bastante bien y porque básicamente es la única opción factible en nuestro caso.

Para poder realizar el envío de correos electrónicos es preciso añadir el código dentro de los propios *endpoints* del proyecto *AppEngine*.

Además, cabe destacar que este servicio requiere la activación del "billing" en la consola de administración de *Google*, es decir, se trata de un servicio de pago.

Cuando se activa esta opción, comienza a contar una cuota gratuita y, cuando ésta es superada, se le pasa la factura al administrador de la aplicación.

A pesar de esto, la aplicación ha sido preparada para su correcto uso ya que el servicio funciona correctamente sin activar la opción de "billing" cuando se está utilizando la base de datos local.

# 5. Pruebas

Se han realizado numerosas pruebas para la verificación del correcto funcionamiento de la aplicación. Todas las pruebas han sido realizadas utilizando tanto la base de datos local como la remota. Todos los pasos han sido verificados en la consola de administración de *Google App Engine*.

Cabe mencionar que siempre que ha sido localizado un problema o algo que podía ser mejorado se han vuelto a reiniciar todas las pruebas desde el principio para verificar que no se han producido efectos laterales durante las modificaciones.

A continuación se describe una de las pruebas realizadas.

1. **Registro de usuario**. Se entra en la pantalla de registro de nuevos usuarios y se introducen los datos del usuario: nombre de usuario “usuario”, contraseña “usuario” y correo electrónico <email>@gmail.com. El usuario es creado correctamente y se vuelve a la pantalla de acceso.
2. **Entrar en la aplicación**. En la pantalla de acceso a la aplicación, se introduce el nombre de usuario “usuario” y la contraseña “usuario”, se pulsa el botón “Recordar” y, por último, el botón “Entrar”. El usuario entra correctamente en la aplicación, que le muestra la lista de los últimos productos introducidos en la base de datos por todos los usuarios.
3. **Buscar un producto y ver sus ofertas**.
    a. **Escenario 1**. Se introduce la letra “t” en la barra de búsqueda de productos y aparece una sugerencia que coincide con el producto buscado. Se pulsa y se muestran sus ofertas.
    b. **Escenario 2**. Se introduce la letra “t” en la barra de búsqueda de productos y aparece una sugerencia. En lugar de pulsar dicha sugerencia, aceptamos la búsqueda y aparecen todos los productos que tienen alguna palabra en su nombre que comience con la letra “t”. Se pulsa sobre el producto deseado y se muestran sus ofertas.
4. **Ver los detalles de una oferta y su localización en el mapa**. Se pulsa sobre una de las ofertas de la lista y se accede a sus detalles. Se muestran todos los campos correctamente (comercio, dirección, etc). Se pulsa sobre el botón “Ver en mapa” y nos muestra en *Google Maps* la dirección donde se encuentra el producto que la ofrece. Navegamos un poco por el mapa y aumentamos y reducimos el nivel de zoom de manera aleatoria.
5. **Añadir un voto y un comentario a una oferta**. Pulsamos el botón “Atrás” y volvemos a los detalles de la oferta.
    a. **Voto**. Se pulsa sobre el botón para añadir un voto negativo y nos pide confirmación, la cual aceptamos. Se añade el voto correctamente y comprobamos que el botón muestra el número anterior al voto más uno. Intentamos votar de nuevo negativamente pero no podemos hacerlo porque ya lo hemos hecho. Intentamos cambiar el voto negativo por un voto positivo pulsando el botón para añadir un voto

positivo y nos pide confirmación, aceptamos y el voto negativo es cambiado por el positivo correctamente. El número de votos negativos decrece en uno y el de votos positivos muestra uno más que antes.

b. **Comentario**. Observamos que el botón "Comentarios" muestra el número correcto de comentarios existentes para la oferta y lo pulsamos. Se muestra correctamente la lista de comentarios existentes para la oferta. Se pulsa el botón "Añadir" de la barra de acciones de *Android* y la aplicación nos muestra una ventana emergente en la que podemos introducir el comentario que queremos añadir. Al terminar se pulsa el botón "Aceptar" y el comentario aparece correctamente junto con el nombre del usuario y la marca temporal en la que ha sido añadido. Se pulsa el botón "Atrás" del dispositivo.

6. **Introducir un producto y una oferta**. Estamos situados en la pantalla de los detalles de la oferta. Se pulsa el botón "Atrás" y volvemos a ver la lista de ofertas. Volvemos a pulsar el botón "Atrás" y nos situamos en la lista de "Productos más recientes". Se pulsa el botón "Añadir" y aparece la pantalla para rellenar los datos específicos del producto. Se rellenan todos los campos y se añade una imagen desde la cámara de fotos. Probamos cambiar la imagen de la cámara de fotos por una de la galería del teléfono satisfactoriamente. Se pulsa el botón "Siguiente" y se muestra la pantalla para introducir los datos específicos de la oferta asociada al producto. Se introducen todos los datos de la oferta y se pulsa el botón "Aceptar". Comprobamos que el producto y la oferta han sido introducidos y asociados correctamente en la base de datos correctamente. La aplicación vuelve a mostrar la pantalla de "Productos más recientes", en la que aparece ya el producto recién introducido. Pulsamos sobre el producto y aparece la oferta introducida. Se pulsa sobre la oferta y vemos sus detalles sin problemas.
7. **Ir al perfil del usuario y borrar el producto introducido**. Volvemos a la lista de productos más recientes pulsando el botón "Atrás" dos veces. Pulsamos la tecla menú del dispositivo y aparecen las opciones del *overflow* de la barra de acciones de *Android*. Se pulsa sobre la opción "Perfil de usuario" y la aplicación muestra el perfil del usuario. Se comprueba que toda la información del usuario es correcta y se pulsa sobre el botón "Productos". Se muestra el único producto que ha añadido el usuario y se pulsa sobre él. Aparecen los campos específicos de ese producto en modo edición. Se pulsa sobre el botón "Eliminar" y se confirma su eliminación. El producto es borrado correctamente, así como todos sus datos asociados (oferta, etc). La aplicación nos muestra la lista de productos del usuario, que ahora ha pasado a estar vacía.

Esta ha sido una de las últimas pruebas realizadas, cuando ya se habían depurado y corregido algunos problemas. Los principales problemas han surgido al probar la herramienta con distintas versiones de *Android*, ya que algunas funciones como, por ejemplo, el acceso a la galería del dispositivo, se implementan de distinta forma.

# 6. Conclusiones y líneas futuras

En el presente *Trabajo de Fin de Grado* se ha desarrollado la aplicación para dispositivos *Android* que cumple con todos los requisitos que se habían establecido en sus comienzos.

La aplicación puede ser utilizada por sus usuarios para intercambiar información acerca del precio de los productos que habitualmente suelen comprar en tiendas físicas. Esto podría suponer un gran ahorro en tiempo y dinero para el usuario. Ahorro económico no sólo por facilitarle el poder encontrar mejores precios de un determinado producto, sino también en gastos de transporte, ya que no necesitaría desplazarse a todos los comercios en los que necesite comprobar el precio de un producto.

Si se cuenta con una base de datos bien poblada, los usuarios podrían ver con muy buenos ojos la aplicación en el momento de probarla y comenzarían a utilizarla. Tener más usuarios repercutiría también directamente en una mayor riqueza de la base de datos, ya que son los propios usuarios los que aportarían la información.

La aplicación tiene un gran potencial que puede ser desarrollado. Pensando en las posibilidades de desarrollo y mejora de la aplicación surgen un buen número de ideas. Por mencionar algunas posibles mejoras referentes a su funcionalidad:

1. **Ruta al comercio**. Ofrecer al usuario la posibilidad de ver en el mapa el camino desde el punto en el que se encuentra hasta donde se encuentra el comercio que quiere visitar.
2. **Distancia al comercio**. Calcular la distancia desde el punto en el que se encuentra el usuario hasta el comercio a visitar. Podría aparecer como información en la lista de ofertas. Debería ser el propio usuario el que decida si activar esta opción o no debido al tráfico que generaría y por motivos de privacidad, ya que la herramienta estaría continuamente consultando dónde se encuentra el usuario.
3. **Comunicación privada entre usuarios**. Posibilidad de que los usuarios se comuniquen entre sí mediante el envío de mensajes privados.
4. **Ver perfil de otro usuario**. Que los usuarios puedan visitar el perfil de otros usuarios para ver qué productos u ofertas han publicado o mandarles un mensaje privado. Se mostraría sólo información no privada y que sea realmente relevante para uso de la herramienta.
5. **Reputación de usuarios**. Poder dar reputación positiva o negativa a otros usuarios para tener una mejor idea sobre la veracidad de las ofertas que publican.
6. **Mejora del perfil de usuario**. Añadir otros elementos que hagan más personalizable la información del usuario. Por ejemplo, mediante la inclusión de una imagen o avatar que identifique al usuario y que él mismo pueda gestionar. La imagen del usuario podría incluso aparecer en la lista de

comentarios publicados o en las propias ofertas. También se podría tener una lista de productos u ofertas favoritos, por ejemplo, el usuario encuentra un producto que le interesa y lo añade a su lista de favoritos que podrá ver desde su perfil de usuario.

7. **Opciones de ordenación**. Que el usuario pueda seleccionar las opciones de ordenación de productos y ofertas. Actualmente se organizan por fecha, primero las más recientes. Estaría bien incluir ordenación por precio, distancia, número de votos, etc.
8. **Gestión de usuarios**. Los administradores podrían desactivar o eliminar cuentas de usuario, hacer administradores a otros usuarios, etc.
9. **Posible apoyo de las empresas**. Ellas mismas serían usuarios VIP o similar y podrían poner sus propios anuncios (servicio de pago).
10. **Estudiar posibles implicaciones legales**. Puede que a algún comercio no le agrade que se publique alguna información.

# Referencias bibliográficas


### Android

http://developer.android.com

http://icons4android.com/

### Google Apps Engine

https://appengine.google.com/

https://console.developers.google.com/

https://cloud.google.com/appengine/docs

https://cloud.google.com/developers/articles/how-to-build-mobile-app-with-app-engine-backend-tutorial/

https://code.google.com/p/objectify-appengine/

### Google Maps

https://developers.google.com/maps/?hl=es

### Eclipse

https://www.eclipse.org/

https://developers.google.com/eclipse/

### JavaMail API

http://www.oracle.com/technetwork/java/javamail/index.html

### General

http://es.wikipedia.org/


### Nota:

Los símbolos pictográficos utilizados de *ARASAAC* (http://catedu.es/arasaac/) son parte de una obra colectiva propiedad de la *Diputación General de Aragón* y han sido creados bajo *Licencia Creative Commons* (*BY-NC-SA*).

# Anexos técnicos

## Manual de usuario

### 1. Pantalla inicial: Datos de acceso a la aplicación

Al iniciar la aplicación se muestra la pantalla para introducir los datos de acceso del usuario.

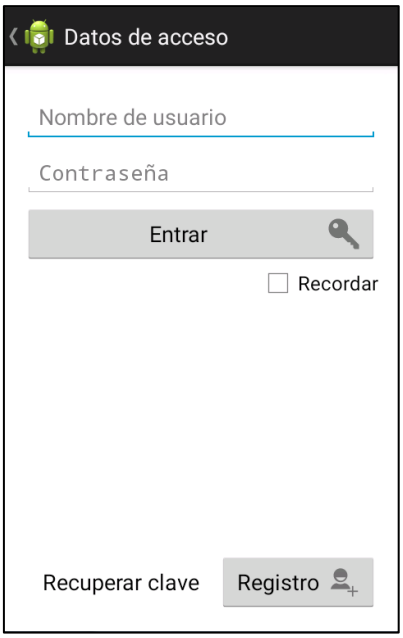


Figura 35. Datos de acceso

### 2. Registrar un nuevo usuario

Para crear un nuevo usuario, desde la pantalla inicial, pulsar el botón “Registro” que aparece en la parte inferior de la figura 35 y aparecerá la pantalla de registro que se muestra en la figura 36.


Figura 36. Registro de usuarios

Para completar el registro de un nuevo usuario habrá que rellenar los siguientes campos:

1. **Nombre de usuario**. El nombre que el usuario desea utilizar para el acceso a la aplicación. Este nombre es único, no pueden existir dos usuarios con el mismo nombre de usuario, si la aplicación avisa de este hecho, habrá que escribir un nombre distinto.
2. **Contraseña**. La contraseña del usuario, se solicitará para poder acceder a la aplicación.
3. **Repetir contraseña**. Se repite la contraseña introducida para verificar que se hizo correctamente.
4. **Email**. Escribir el correo electrónico del usuario. El email es único, no pueden existir dos usuarios con el mismo email, si la aplicación avisa de este hecho, habrá que escribir un correo electrónico diferente.

Por último se pulsará el botón “Aceptar” para finalizar el registro. La aplicación volverá a la pantalla inicial.

## 3. Acceso a la aplicación

En la pantalla inicial, rellenar los campos “Nombre de usuario” y “Contraseña” con los datos de un usuario que ha debido ser previamente registrado. Pulsar el botón “Entrar” para finalizar.

Se pueden recordar los datos de acceso a la aplicación para evitar tener que escribirlos en cada inicio de la misma, se debe marcar la casilla “Recordar” antes de pulsar el botón “Entrar”.

## 4. Recuperar clave

Para recuperar los datos de acceso a la aplicación de un usuario, pulsar sobre el texto “Recuperar clave” situado en la parte inferior de la pantalla inicial. Aparecerá el diálogo emergente que se muestra en la siguiente figura.

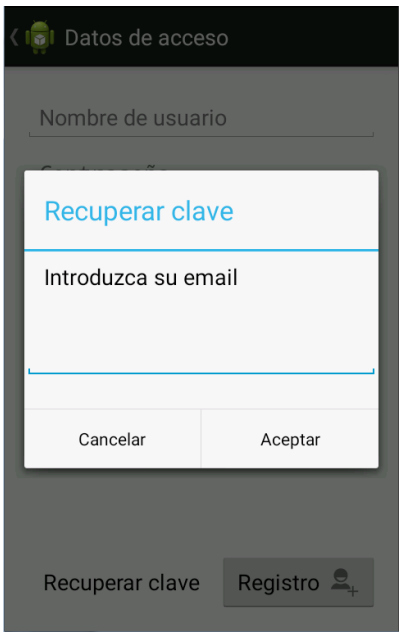


Figura 37. Recuperar clave

Rellenar el campo indicado como “Introduzca su email” con el correo electrónico del usuario. La aplicación enviará un correo electrónico al usuario con sus datos de acceso (nombre de usuario y contraseña).

## 5. Lista de productos más recientes.

La pantalla que muestra la lista de productos más recientes es la que aparece justo después de acceder a la aplicación.


Figura 38. Productos más recientes

## 6. Buscar un producto.

Para buscar un producto, se debe estar viendo la lista de productos más recientes o haber realizado ya alguna búsqueda. Pulsar el botón "Buscar" de la barra de acciones superior que aparece marcado en la siguiente figura.

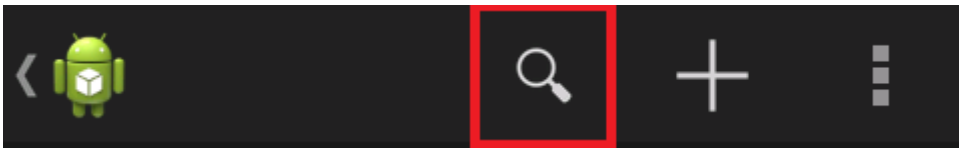

Figura 39. Barra de acciones: Buscar

Tras pulsar este botón se expandirá la barra de búsqueda y se podrá escribir el nombre del producto a buscar dentro del campo de texto.

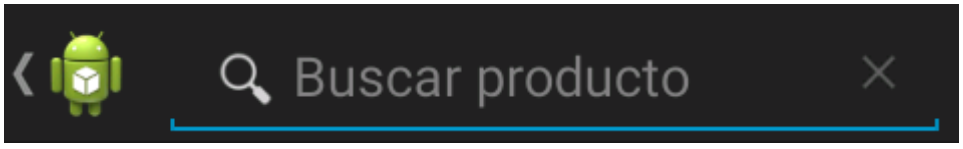


Figura 40. Barra de búsqueda

Una vez que se ha terminado de escribir el nombre del producto hay que aceptar la búsqueda y aparecerá una lista de productos que contendrá el producto o productos sugeridos que más se parecen al nombre introducido.

Si durante la escritura del texto aparece debajo una lista de sugerencias de productos coincidentes y el producto que se está buscando es uno de ellos, se puede pulsar directamente sobre él y nos llevará a su lista de ofertas.


Figura 41. Barra de búsqueda: Sugerencias

## 7. Ver ofertas de un producto

Para ver la lista de ofertas de un producto se debe pulsar sobre él en la lista de productos o en la lista de sugerencias que aparece al buscar un producto. La siguiente figura muestra la lista de ofertas de un producto.


Figura 42. Lista de ofertas

## 8. Ver detalles de una oferta

Se puede ver toda la información referente a una oferta pulsando sobre una de ellas en la lista de ofertas. La siguiente figura muestra algunos detalles de una oferta, el resto de detalles se puede ver deslizando hacia abajo la pantalla.


Figura 43. Detalles de una oferta I

## 9. Ver en un mapa la dirección en la que se encuentra una oferta

Estando situados en la pantalla que muestra los detalles de una oferta, pulsar el botón “Ver en mapa”, que aparece en la figura 44.

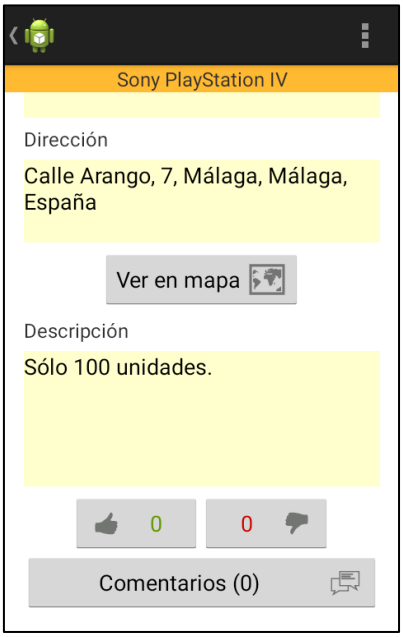


Figura 44. Detalles de una oferta II

## 10. Votar una oferta

Se puede votar una oferta positiva o negativamente pulsando sobre los botones que pueden observarse en las figuras 44 y 45.


Figura 45. Votos

Aparecerá un mensaje de confirmación antes de añadir el voto. Sólo se permite un voto por usuario en cada oferta. El usuario puede cambiar su voto en cualquier momento.

## 11. Ver los comentarios de los usuarios sobre una oferta

Para ver los comentarios que los usuarios han añadido a las ofertas, presionar el botón "Comentarios" que aparece en la figura 44. Este botón muestra el número de comentarios que ha recibido la oferta por parte de los usuarios.

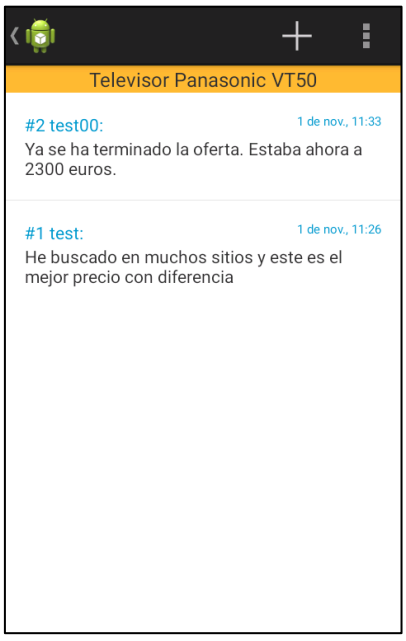


Figura 46. Comentarios de los usuarios

El texto del comentario aparece junto con el número de comentario, el usuario que lo escribió y la fecha en la que fue añadido.

## 12. Añadir un comentario a una oferta

Los usuarios pueden añadir todos los comentarios que deseen sobre una determinada oferta. Para ello se debe estar situado en la pantalla que muestra todos los comentarios sobre una determinada oferta y que aparece en la figura 46. Una vez ahí, pulsar el botón "Añadir" que aparece marcado en la siguiente figura.

Figura 47. Barra de acciones: Añadir

Aparecerá un diálogo emergente en el que se puede introducir el texto del comentario. Una vez se ha terminado de escribir, pulsar el botón "Aceptar".

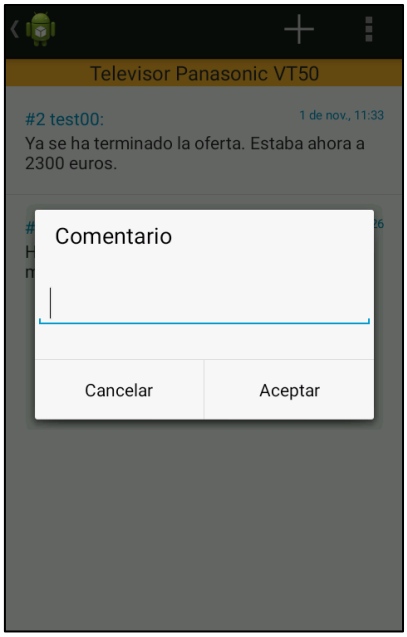


Figura 48. Escribir comentario

## 13. Refrescar la lista de productos, ofertas o comentarios.

En cualquier momento, otro usuario puede haber introducido o modificado un producto u oferta o haber escrito un nuevo comentario. Si la lista de elementos que se muestra en pantalla ha sido actualizada antes de que ese hecho se produzca, los nuevos datos no se mostrarán en pantalla. Para ello, se puede pulsar el botón

"Refrescar", que mostrará la lista de elementos más actual. Este botón está situado en la barra superior de acciones de la aplicación.

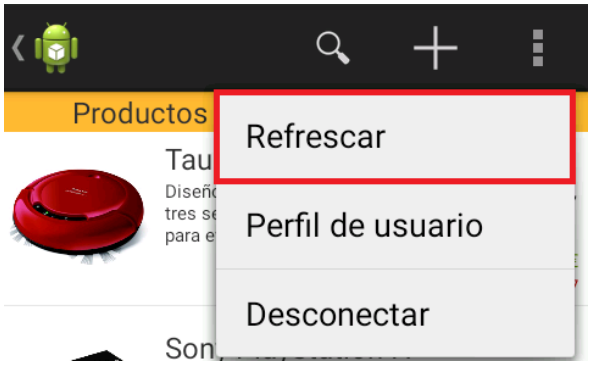


Figura 49. Barra de acciones: Refrescar

El botón "Refrescar" aparecerá siempre que estemos viendo una lista de productos, ofertas o comentarios y no estemos dentro del perfil del usuario.

## 14. Añadir un producto

Para añadir un nuevo producto se debe estar viendo la lista de productos más recientes o haber realizado ya alguna búsqueda. Pulsar el botón "Añadir" mostrado en la Figura 47 y aparecerá la siguiente pantalla.

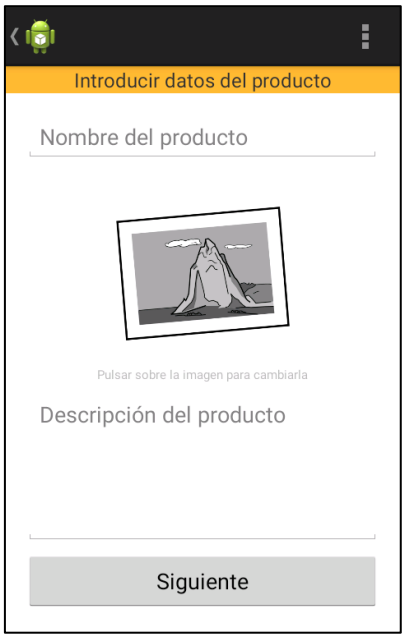


Figura 50. Introducir datos del producto

Introducir el nombre del producto y su descripción. Esta descripción se refiere a las características propias del producto, la descripción o comentarios sobre la oferta se realizará más adelante.

Para añadir una imagen al producto, pulsar sobre la imagen que aparece en el centro de la pantalla y aparecerá un mensaje para seleccionar si se quiere elegir una imagen desde la propia galería del dispositivo o directamente tomar una foto al producto.


Figura 51. Opciones de selección de imagen

Tras seleccionar la imagen, ésta aparecerá en el centro de la pantalla de la figura 50, sustituyendo la imagen genérica que había anteriormente.

Cuando se terminen de rellenar todos los campos, pulsar el botón siguiente y aparecerá la pantalla que permitirá rellenar los datos de la oferta.


Figura 52. Introducir datos de la oferta

Los campos de la oferta son:

1. **Comercio.** Nombre del comercio en el que se encuentra la oferta del producto.
2. **Dirección del comercio.** Hay dos formas de introducirla.
    a. **Escribiendo la dirección**. El nombre de la calle y el número es suficiente. La dirección será buscada en *Google Maps* y añadida adecuadamente con todos sus datos, latitud y longitud incluidos. Para que esto sea posible, hay que seleccionar adecuadamente la provincia y localidad.
    b. **Pulsando un botón**. Si estamos físicamente en el comercio que ofrece el producto que se está añadiendo, basta con pulsar el botón situado a la derecha del campo "Dirección del comercio". La aplicación añadirá el texto de la dirección automáticamente al campo "Dirección del comercio" sin tener que escribirla a mano. Para poder usar esta función, la opción para permitir el acceso a tu ubicación del dispositivo *Android* debe estar activada. Si no lo está, la aplicación preguntará si se desea activarla y, en caso de ser así, mostrará directamente el menú de *Android* que permite su activación.
3. **Provincia**. Provincia en la que se encuentra la oferta.
4. **Localidad**. Localidad en la que se encuentra la oferta.
5. **Precio**. Precio total (todas las unidades) de la oferta en euros.
6. **Unidades**. Número de unidades del producto que entran en la oferta.
7. **Comentarios sobre la oferta**. Comentarios que desea hacer el usuario acerca de la oferta. No confundir con los comentarios que aparecen en la lista de comentarios de todos los usuarios. Este comentario aparecerá como información extra en los detalles de la oferta.

Por último, deslizar la pantalla de la figura 52 hasta abajo del todo para ver el botón "Aceptar", pulsarlo y el producto junto con su oferta se añadirán a la aplicación. La aplicación muestra la pantalla de productos más recientes, que ya contendrá el producto recién introducido.

## 15. Añadir una oferta

Para añadir una oferta a un producto que ya existe, se debe seleccionar dicho producto para que se muestre su lista de ofertas. Una vez hecho esto, pulsar el botón "Añadir" que aparece marcado en la figura 47. La aplicación mostrará el formulario para rellenar los datos de la oferta tal como se muestra en la figura 52. Los datos a rellenar ya han sido descritos en el apartado 14. Una vez introducidos todos los datos de la oferta, pulsar el botón "Aceptar" y la oferta será añadida al producto. La aplicación mostrará la lista de productos más recientes, en la que aparecerá el producto al que le hemos añadido la oferta.

## 16. Acceso al perfil de usuario

Para acceder al perfil del usuario que ha accedido a la aplicación, se debe estar viendo la lista de productos más recientes o haber realizado ya alguna búsqueda. La siguiente figura muestra una parte del perfil del usuario.


Figura 53. Perfil de usuario: Datos personales

## 17. Modificar datos personales

Para modificar los datos personales del usuario hay que ir al perfil de usuario (mostrado en la figura 53) y editar los campos que aparecen en el apartado “Datos personales”.

1. **Nombre completo**. El nombre completo del usuario. No confundir con el nombre de usuario, se trata del nombre real de la persona. Este campo no es obligatorio.
2. **Email.** Se puede modificar el email del usuario por otro que no exista ya en la aplicación.
3. **Contraseña**. Para modificar la contraseña se debe introducir la contraseña actual y la nueva contraseña que se quiere utilizar.

Una vez modificados los datos se pulsará el botón “Aplicar” que aparece justo debajo para que se guarden los cambios realizados. En el caso de la contraseña, se verificará que la contraseña actual es correcta antes de establecer la nueva contraseña, en caso de no serlo se avisará al usuario.

## 18. Seleccionar la ubicación de búsqueda de productos

Se puede establecer la zona en la que el usuario desea buscar los productos. Esto puede hacerse desde el perfil de usuario en los apartados “Dónde buscar” y “Buscar ofertas a nivel”, que se muestran en la figura 54.

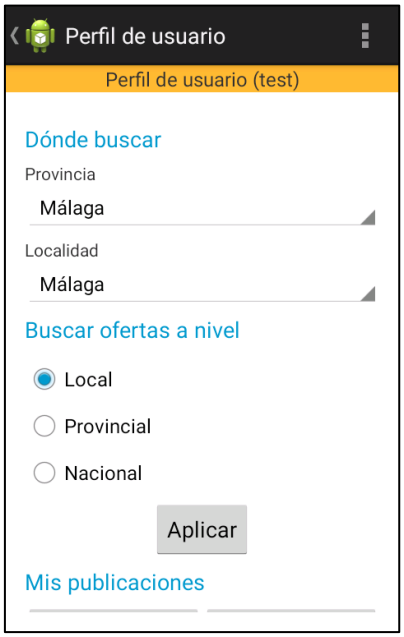


Figura 54. Perfil de usuario: Ubicación de búsqueda

1. **Dónde buscar**. Se seleccionará la provincia y la localidad en la que se desean buscar los productos.
2. **Buscar ofertas a nivel**.
    a. **Local**. Se buscará únicamente en la localidad seleccionada en el apartado “Dónde buscar”.
    b. **Provincial.** Se buscará en todas las localidades de la provincia seleccionada en el apartado “Dónde buscar”, independientemente de la localidad que se tenga seleccionada.
    c. **Nacional**. Se buscará en todas las provincias y en todas las localidades. Si se selecciona esta opción no es necesario rellenar los campos del apartado “Dónde buscar”.

Una vez seleccionadas las opciones de búsqueda, pulsar el botón aplicar que se encuentra justo debajo para aplicar los cambios.

## 19. Editar o borrar un producto

Si el usuario que ha accedido a la aplicación ha añadido algún producto a la aplicación, puede editarlo desde su perfil de usuario. Para hacerlo, pulsar el botón “Productos” del apartado “Mis publicaciones” que se encuentra en el perfil de usuario.

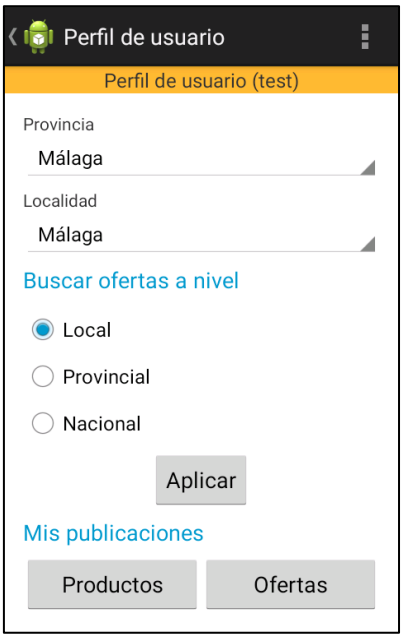


Figura 55. Perfil de usuario: Mis publicaciones

Aparecerá la lista de productos que el usuario ha añadido. Pulsar sobre el producto que se desea editar y aparecerá la pantalla de modificación de productos.


Figura 56. Editar producto

Modificar los datos tal como se explicó en el apartado 14 y pulsar el botón “Aplicar” para guardar los cambios. La aplicación volverá a la lista de productos del usuario.

Para borrar un producto, en la pantalla de modificación de productos del perfil de usuario, pulsar el botón “Borrar” y confirmar. El producto será borrado junto con toda su información asociada (ofertas, comentarios, etc).

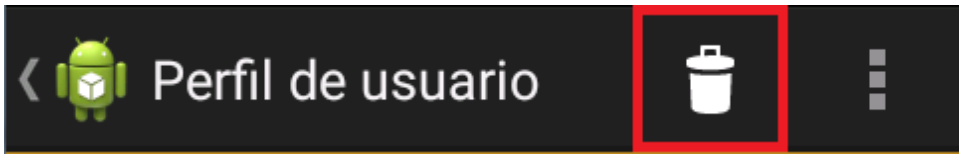


Figura 57. Barra de acciones: Eliminar

## 20. Editar o borrar una oferta

Si el usuario que ha accedido a la aplicación ha añadido alguna oferta a la aplicación, puede editarla desde su perfil de usuario. Para hacerlo, pulsar el botón “Ofertas” del apartado “Mis publicaciones” que se encuentra en el perfil de usuario y que aparece en la figura 55.  Aparecerá la lista de ofertas que el usuario ha añadido.


Figura 58. Lista de ofertas del usuario

Pulsar sobre la oferta que se desea editar y aparecerá la pantalla de modificación de ofertas.

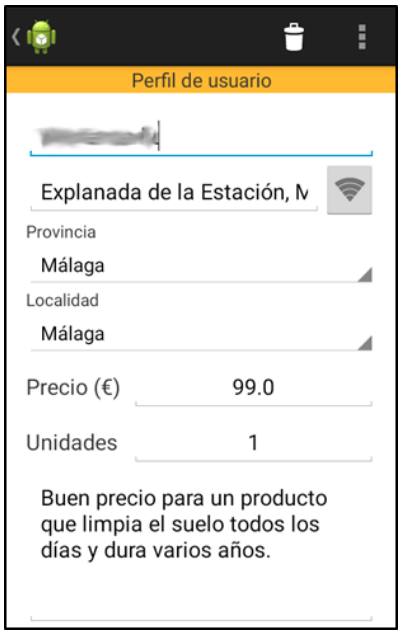


Figura 59. Editar oferta

Modificar los datos de la oferta teniendo en cuenta lo que se explicó en el apartado 14 y, una vez terminado, deslizar la pantalla hacia abajo hasta ver el botón “Aplicar” y pulsar sobre él. La oferta quedará modificada y la aplicación mostrará de nuevo la lista de ofertas del usuario.

Para borrar una oferta, en la pantalla de modificación de ofertas del perfil de usuario, pulsar el botón “Borrar” (ver figura 57) y confirmar. La oferta será borrada junto con toda su información asociada (comentarios, votos, etc).

## 21. Desconectarse

En cualquier momento se puede desconectar de la aplicación para volver a la pantalla inicial de acceso a la aplicación pulsando el botón “Desconectar” situado en la barra de acciones superior. Confirmar para terminar.


Figura 60. Barra de acciones: Desconectar

## 22. Salir de la aplicación

Para salir de la aplicación, pulsar el botón “Atrás” del dispositivo o el situado en la barra de acciones superior hasta llegar a la pantalla que muestra los productos más recientes. Una vez ahí, pulsar una vez más el botón “Atrás” y confirmar para salir de la aplicación.

Figura 61. Barra de acciones: Atrás