Trabajo Final de Carrera

Ingeniería Informática de Sistemas (UOC)

Arcadio Carballares Martín
-2014-

Análisis Funcional y Técnico

v1.0.02 - 3 de enero de 2013

Índice

1. Introducción

Los dispositivos **móviles se han popularizado progresivamente** en el mercado informático y poco a poco han reemplazado muchas de las funciones que tradicionalmente se reservaban a los ordenadores personales. El consumo doméstico de estos dispositivos se ha disparado y las empresas de software se han visto en la necesidad (y en ocasiones la oportunidad) de **trasladar parte de sus desarrollos a estos entornos *mobile*** para ampliar o complementar servicios que ya ofrecían.
Actualmente cualquier desarrollo de software parece obligado a entregar también una versión adaptada a los dispositivos móviles si quiere penetrar con cierta garantía en el mercado de los usuarios domésticos o profesionales. Este requisito ha obligado a **readaptar los paradigmas de la ingeniería** en áreas como **GUI** principalmente debido a que los nuevos aparatos responden a unas necesidades (pantalla, usabilidad, etc.) y unos estándares diferentes a los tradicionales, como el de las pantallas de ordenador. Pero no sólo se ha producido una evolución en los sistemas de interfaz sino que los programadores deben emplear **nuevas técnicas más apropiadas** que solucionen cuestiones de rendimiento, fiabilidad, versatilidad, sistema, distribución… y sobre todo adaptarse a una nueva forma de concebir la producción de software (App's store). Como resultado se han generado numerosos ecosistemas de desarrollo (Android, iOS, BB, web móvil, etc.) que obligan a una cierta **especialización** y **caracterización** de cada aplicación.

Las teorías de desarrollo de software, ya de por sí novedosas comparadas con otras ramas científicas, están **readaptando conceptos y metodologías** afianzadas para dar cabida a estas nuevas formas de programación condicionadas por las necesidades del mercado y la, cada vez más intensa participación del usuario final en el mundo del software.

El objetivo de esta TFC es **proporcionar el análisis, diseño e implementación de una aplicación para entornos móviles** y adaptar la metodología clásica de ingeniería de software a las plataformas *mobile*.

2. Descripción

GeoPost-it: Geo-posicionamiento de notas informativa es una aplicación *mobile* para insertar notas privadas y públicas basadas en la posición del dispositivo.

La aplicación está diseñada para almacenar una serie de datos que se distribuyen en puntos localizables físicamente por GPS (o por cualquier otro sistema GIS) y que se muestran una vez que se alcanza cada posición de manera que sirva como utilidad para vincular la información deseada a un espacio geográfico concreto, accesible para el aprovechamiento de los usuarios.

Las **funciones y capacidades** que proporciona son:

- Gestión de **información geo-referenciada**.
- **Filtrado por posición**, el dispositivo sólo recibe la información en su radio de acción dentro de unos límites establecidos.
- **Notificaciones a usuarios** sobre marcas nuevas o actualizadas en el radio de acción del dispositivo basadas en el sistema Google Cloud Messaging (GCM) de Google para Android.

El usuario de móvil puede registrar su posición actual añadiendo una marca con nota asociada a un lugar físico. Más adelante, cuando el mismo dispositivo alcanza la posición con marcas la aplicación puede leerlas. Es indispensable el **desplazamiento físico** (o la ubicación física) para que la lectura de la información se lleve a cabo.

El **objetivo es compartir datos de utilidad y recordatorios** en ubicaciones físicas que la aplicación notifica en el momento de coincidencia espacial.

GeoPos-it está compuesto por:

- Una **aplicación Android** para las operaciones de entrada y lectura
- Una **plataforma de soporte** (back-end) para el almacenamiento y gestión de los datos compartidos por otros usuarios.

GeoPost-it es el resultado de un esfuerzo por demostrar las capacidades GIS de los dispositivos Android y de las posibilidades que ofrecen estas herramientas para proporcionar sistemas de geo-posicionamiento útiles para los usuarios .

3. Metodología

La metodología escogida se basa en **Scrum**, dado el carácter novedoso de las tecnologías utilizadas que no se rigen por un patrón clásico y el solapamiento de perfiles (cliente, desarrollador).

Scrum establece un sistema de trabajo basado en un proceso iterativo e incremental con en entornos de *trabajo ágiles* para potenciar la rapidez y versatilidad del desarrollo.

La metodología propuesta, además, introduce un proceso de generación continua de prototipos y de *releases* que implementan incrementalmente cada funcionalidad o requisito, de tal manera que en todo momento se disponga de una versión operativa limitada a las capacidades descritas para cada entrega.
Scrum establece la transversalidad de roles y en este sentido se adapta al modelo escogido para asumir todos los perfiles necesarios del proyecto.

Algunos de los **puntos asumidos para la metodología**:

- Potenciar el desarrollo y el resultado sobre otros aspectos: el análisis, la documentación, las pruebas deben estar al servicio del objetivo de cada reléase y no entorpecer (sobre-documentación, parálisis por análisis, etc.) el flujo de desarrollo.
- Se plantean equipos auto-organizados con capacidades para asumir diferentes roles y tener una perspectiva global del proyecto para asumir sin problemas cada rol necesario en cada momento.
- El desarrollo es tolerante al cambio de requisitos, de manera que el clientes pueden cambiar de idea sobre lo que quiere y necesita y el equipo de desarrollo debe reflejar en la siguiente *release* las nuevas funcionalidades acordadas.
- Maximizar la capacidad del equipo en cumplir con las entregas establecidas en el menor tiempo posible y responder y solucionar los imprevistos que se produzcan en el proceso.
- Trabajo por hitos (sprint) con objetivo de entregables definidos en tiempos cortos de trabajo (1 semana).

4. Análisis de Requisitos

4.1. Requisitos funcionales

- Geo-posicionar marcas con notas textuales: El usuario debe poder hacer una marca (con nota) sobre la posición actual de manera sencilla y utilizando algún sistema de ubicación del teléfono (Wifi, GPS, etc.).
- Almacenar notas: El usuario debe poder almacenar en un registro interno del móvil las notas que ha vinculado a la posición marcada previamente. Estas notas deben tener un sistema para gestionarlas.
- Compartir notas: El usuario debe poder establecer la visibilidad de las notas como públicas para poder compartirlas mediante un sistema de acceso público por otros usuarios de la aplicación.
- Notificaciones de evento: La aplicación dispondrá de un sistema que notifique y avise al usuario cuando está dentro de la zona de una marca. Esta zona estará delimitada por un radio de distancia suficiente para el correcto funcionamiento de la aplicación.

4.2. Requisitos no funcionales

- La aplicación necesita permisos para usar posicionamiento por GPS y/o por Redes.
- La aplicación debe funcionar en la mayor variedad de versiones de Android, aunque principalmente esté focalizada en las más recientes.
- La información de la aplicación debe visualizarse correctamente en la mayoría de las pantallas de los dispositivos del mercado, tanto móviles como tabletas.
- Es necesario un hosting con tecnología java para almacenar la aplicación web que servirá los datos públicos a la aplicación. En este caso se opta por una solución Cloud como **Heroku** o **CloudBees**.

5. Análisis Funcional

5.1. Funcionalidades

- Gestionar marca: Añadir, editar y borrar una marca con información textual.
- Notas públicas y privadas: Al crear una nota se puede almacenar en el dispositivo (privada) o enviarla a internet para dominio público (pública).
- Visualizar mapa con las marcas posicionadas.
- Notificación de marca alcanzada: La aplicación muestra una alerta cuando el dispositivo se encuentra a una determinada distancia (1 kilómetro por defecto) de una marca indicando la posición (zona de notificación).
- Notificación de marca creada o actualizada: La aplicación notifica las marcas que se actualizan y/o crean dentro del radio de acción (1 kilómetro por defecto) de la posición actual del dispositivo de manera que el usuario siempre está informado de los comentarios que se producen a su alrededor.

5.2. Casos de uso

- **Gestión de marca** (alta, baja y modificación)

Identificador	C-01
Ámbito	Sistema local
Actor	Usuario
Descripción	Gestionar el mantenimiento de los post-it que almacena el sistema. Alta, baja y modificación.
Objetivo	Geo-localizar mensajes
Precondición	GPS o sistema GIS activado en el dispositivo. Conexión a Internet.
Escenario	Es sistema devuelve una alerta de operación

normal	realizada correcta o incorrectamente.
Escenario alternativo	No es posible captar señal de geo-localización y la operativa no continúa.

- **Lectura de marca**: El usuario se desplaza hasta la nota ubicada en el mapa a una distancia mínima para que la aplicación responda al geo-posicionamiento.

Identificador	C-02
Ámbito	Sistema local
Actor	Usuario
Descripción	El usuario lee un post-it que está accesible en la aplicación.
Objetivo	Lectura de información.
Precondición	GPS o sistema GIS activado en el dispositivo. Conexión a Internet. La aplicación ha localizado una marca y el dispositivo está en la zona de alcance física para poder visualizarla.
Escenario normal	La lectura de la información se realiza correctamente en pantalla.
Escenario alternativo	Antes de la lectura se pierde la referencia posicional y la aplicación bloquea el acceso por falta de datos.

- **Notificación de marca alcanzada**

Identificador	C-03
Ámbito	Sistema local y remoto
Actor	Aplicación
Descripción	La aplicación rastrea las marcas asociadas a la posición actual del dispositivo y si encuentra registros avisa al usuario automáticamente de la existencia de post-it en la zona de acción.
Objetivo	El usuario dispone de un servicio automático de notificación sobre marcas privadas y públicas.
Precondición	GPS o sistema GIS activado en el dispositivo. Conexión a Internet para la visualización de los mapas.
Escenario normal	El dispositivo rastrea y encuentra las marcas y muestra una pantalla emergente para avisar de su existencia.
Escenario	Falla la conexión con el sistema remoto y sólo se

alternativo	muestran las marcas locales. Fallo del sistema de geo-localización del dispositivo y no se notifican las marcas de la zona.

- **Compartir marca**: El usuario crea una nota en una posición y la hace pública para que otros usuarios de la aplicación accedan a la información vinculada a ese mismo punto físico.

Identificador	C-04
Ámbito	Sistema local
Actor	Usuario
Descripción	Generar un post-it público para que pueda ser rastreado por otros dispositivos.
Objetivo	Compartir marcas con otros usuarios.
Precondición	GPS o sistema GIS activado en el dispositivo. Conexión a Internet.
Escenario normal	Es sistema devuelve una alerta de operación realizada correcta o incorrectamente.
Escenario alternativo	No es posible captar señal de geo-localización y la operativa no continúa.

- **Visualización de marca en el mapa**: La aplicación muestra una pantalla con un mapa en donde se ubican geográficamente las marcas.

Identificador	C-05
Ámbito	Sistema local
Actor	Usuario
Descripción	Mostrar la posición de la marca en Google Maps. El usuario puede ampliar la información que contiene seleccionando cada una de las marcas dentro de la zona de interacción.
Objetivo	Geo-posicionar las marcas cercanas a un espacio físico gráficamente en un mapa.
Precondición	GPS o sistema GIS activado en el dispositivo. Conexión a Internet para la visualización de los mapas.
Escenario normal	Muestra un mapa de Google con los típicos pin señalando la posición de cada marca.
Escenario alternativo	Falla la conexión con Google Maps. Fallo del sistema de geo-localización del dispositivo y no se muestran las marcas remotas en el mapa (sólo las marcas almacenadas en el sistema local).

5.3. Prueba de concepto

Las pruebas de concepto nos permite abordar soluciones parciales que garanticen la **viabilidad de la solución final**.

- Google Maps: Registrar API v2 y añadir componentes a la pantalla basados en GIS. Esto incluye reproducir la lógica para geo-posicionar la localización actual y centrarla en el mapa obteniendo los datos del GPS del dispositivo
- Incluir Push notifications con Google Cloud Messaging for Android de manera que cada vez que se registre o se actualice una nueva marca se propague una alerta a todos los dispositivos registrados y que se encuentren en el radio de acción.
- Mostrar información de marca: Elegir una post-it y mostrar su información.
- Mostrar una notificación al pasar por una posición. Esta opción incluye crear un servicio *background* con un evento de activación para alertar al usuario.
- Obtener datos de back-office: Conectar al sistema back-end vía REST y obtener datos del sistema de apoyo remoto.

6. Diseño Técnico

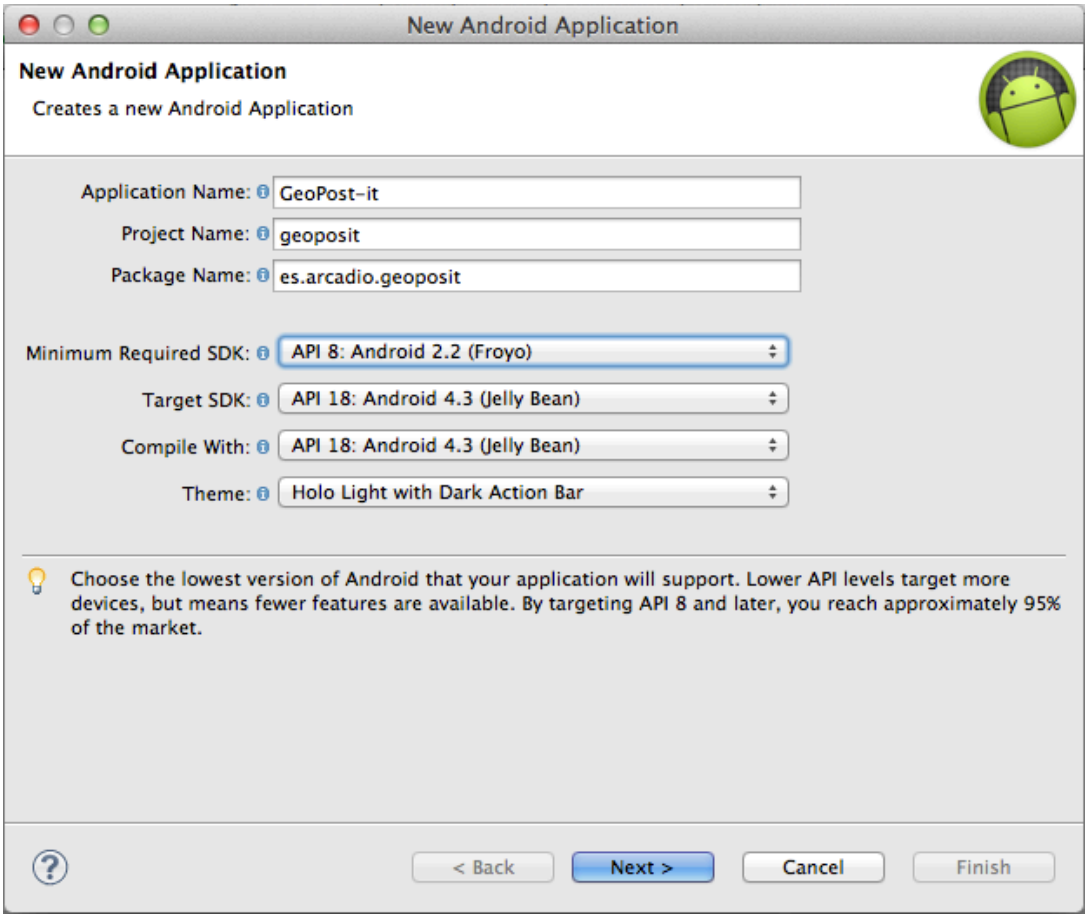
6.1. Tecnología y Herramientas para el desarrollo

- **Java Android** con Android Developer Tools (ADT).
- **GPS** para la obtención de coordenadas.
- **Google Maps Android API v2** para el sistema cartográfico.
- **Google Cloud Messaging** for Android para las notificaciones.
- **PostgreSQL** para los servicios web de almacenamiento en remoto**.**
- **Spring MVC-Hibernate** con **REST**
- **Servicios WEB** con entrada y salida en formato **JSON.**

6.2. Requisitos

La aplicación está soportada para dispositivos móviles con la **android v4.1.2 SDK 16 - Jelly Bean** o superior, con la opción de GPS activado preferiblemente y conpermisos y/o soporte para:

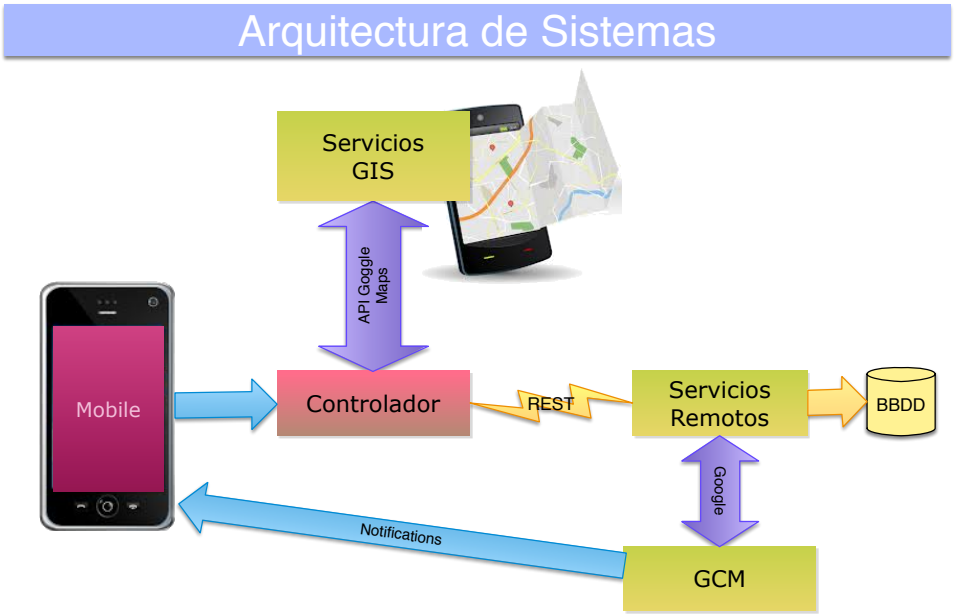
- Google Play Services
- Google Maps v2
- Google Cloud Messaging for Android
- Conexión a internet o WIFI
- Aplicación Google Play Store instalada



6.3. Arquitectura

La aplicación consta de:

- Sistema cliente, basado en la tecnología de Android
- Sistema Back-End, para los servicios de gestión remota de datos basada en Spring MVC.
- Sistema Cloud para mensajería basada en el GCM de Google para Android.



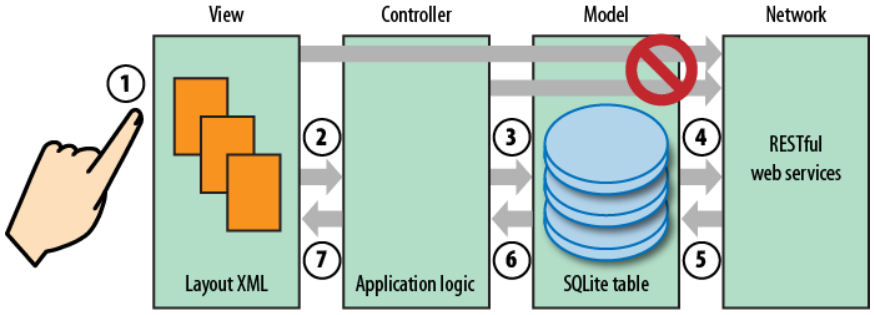
6.4. Cliente Android

El sistema cliente es una **aplicación Android** que utiliza la API REST de Android para las comunicaciones remotas:

- **API REST** de Google. REST (Representational State Transfer) se asienta sobre el protocolo http para el trasporte de información entre el cliente y el servidor utilizando llamadas asíncronas (Android Asynchronous HTTP Client API).
- **Librería JASON** para la gestión de datos en JSON permite la codificación y decodificación de mensajes livianos en un formato estándar y generalizado para el intercambio de datos. Esta librería ofrece funcionalidades para la conversión entre objetos Javascript y objetos Java y ofrece un mecanismo sencillo y eficaz para las operaciones CRUD de creación y actualización de datos entre el cliente y el servidor.
- **Sistema de peticiones HTTP** sobre la base del paradigma Cliente-Servidor y el **patrón MVC** basado en vistas y controladores de Spring.

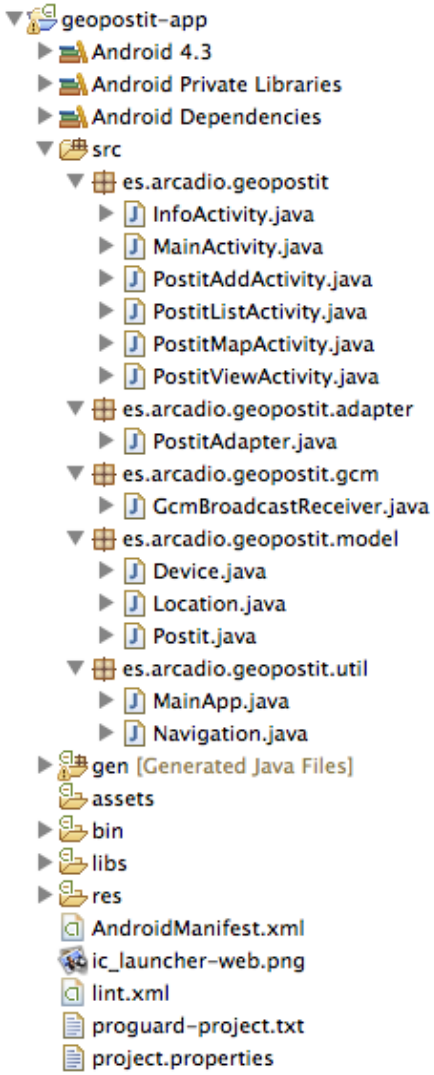
La aplicación se conecta con el sistema de back-end haciendo peticiones que son gestionados por los **controladores de Spring** que se encargan de

despachar las operaciones implicadas en la obtención de los datos a través de **servicios** y **DAO's** (en este caso a través de EntityManager de la implementación de JPA de Hibernate). Una vez realizadas las operaciones en la parte servidora los datos son convertidos a formatos de texto JSON para devolverlos al cliente de forma asíncrona.



Step 1: View - user input.
Step 2: Controller listens for events.
Step 3: Controller invokes a managed query on the content provider (aka the model).
Step 4: The content provider invokes a RESTful YouTube request and prepares to receive results.
Step 5: YouTube responds and the parser inserts results in the content provider SQL cache.
Step 6: The provider notifies the View and Controller.
Step 7: The View updates.

Las interfaces gráficas (o pantallas) son clases que heredan de la clase **Activity de Android**. Por cada pantalla se implementa una interfaz **Navigation** para el **sistema de navegación** de la aplicación.
La estructura del proyecto Android mantiene las directrices generales para el desarrollo de aplicaciones móviles establecida por Google y tal y como se generan a partir del ADT.



En la descripción general del proyecto se incluyen los manifiestos obligatorios para obtener **permisos y servicios necesarios** para el funcionamiento de la aplicación:

- **Permisos de usuario para Internet y Localización**
  - android.permission.INTERNET
  - android.permission.ACCESS_WIFI_STATE
  - android.permission.ACCESS_NETWORK_STATE
  - android.permission.ACCESS_COARSE_LOCATION
  - android.permission.ACCESS_FINE_LOCATION
- **Servicio de notificaciones con GCM**
  - android.permission.GET_ACCOUNTS
  - android.permission.WAKE_LOCK
  - com.google.android.c2dm.permission.RECEIVE
- **Servicio de Mapas Maps API**
  - com.google.android.maps.v2.API_KEY

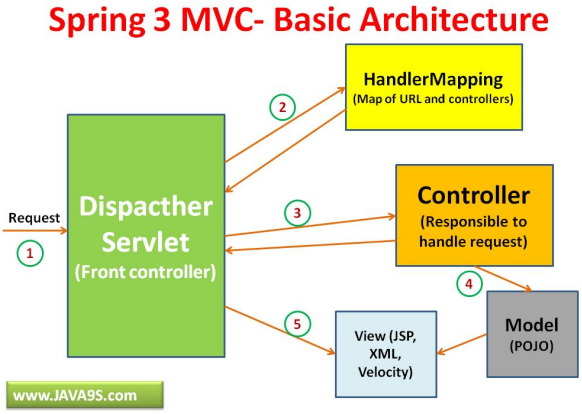
6.5. Sistema Back-End

El sistema back-end es la parte que **gestiona los datos en el servidor** e interactúa con la base de datos **PostgreSQL** bajo el entorno Cloud de **Heroku**.

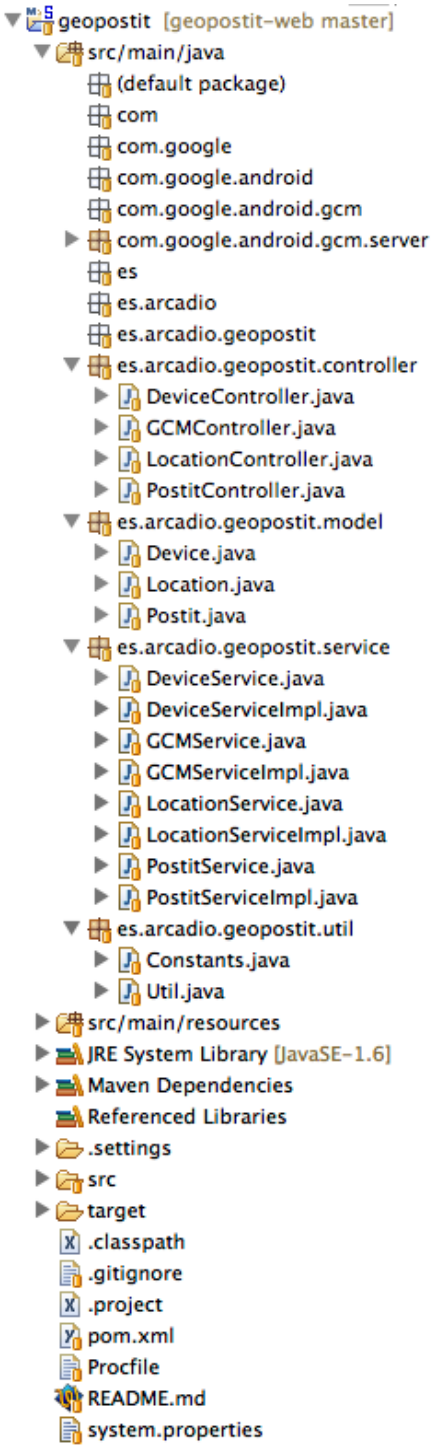
A través de una estructura basada en **Spring MVC** la aplicación Android cliente obtiene los datos necesarios para realizar todas las operaciones que se proporcionan al usuario.

6.5.1. Spring MVC

La comunidad de Spring creó un subproyecto para desarrollar un sistema basado en el **patrón Model-View-Controller (MVC)**. El resultado fue un framework sencillo, con soporte integral para este patrón de software y que aprovecha muchas de las características de Spring, incluida la Inversion of Control (IoC) y la Inyección de Dependencias.



La **arquitectura de Spring MVC** está diseñada para responder a las peticiones HTTP mediante un flujo en el que intervienen diferentes módulos. La fachada despacha las peticiones al sistema mediante la resolución de la ubicación de los recursos (mapping) hasta el controlador que es el encargado de gestionar los servicios necesarios para componer la respuesta dentro de un modelo datos que se aplica sobre una concreta.



El proyecto ha sido generado utilizando las plantillas de Spring proporcionadas por el **servicio de Heroku** que incluye la estructura de una **arquitectura J2EE**:

- **Capa Cliente** para la administración web y un sistema basado en JSON para las vistas que consume la aplicación Android.
- **Capa de Negocio** basada en los servicios invocados desde cada controlador y el sistema GCM de Google.
- Capa de **Persistencia** con JPA-Hibernate.

6.5.2. Heroku Cloud

Heroku es una de las primeras **Plataforma como Servicio (PasS)** que soporta JAVA (y otros lenguajes de programación) y ofrece una infraestructura en Cloud (en la Nube) de forma gratuita (para cuentas básicas).

Algunas de las **características** que ofrece son:

- Soporte gratuito a diferentes lenguajes, entre los que destaca JAVA
- Gestión de código basado en Git.
- Consola de control remoto.
- Servicios basados en el Cloud de Amazon Web Services.
- Basado en contenedores ligeros o **dynos** fácilmente escalables.
- Otros servicios como Bases de Datos, Monitorización, NoSQL, etc.

Geopost-it aprovecha las capacidades de Heroku para albergar la parte servidora de la aplicación móvil y aprovechar las capacidades de escalabilidad que permiten adaptarse a la demanda del servicio.
El código fuente de esta parte está, igualmente, gestionado por el servicio de repositorios basados en la **tecnología Git** que es una excelente herramienta SCM (Source Code Management).

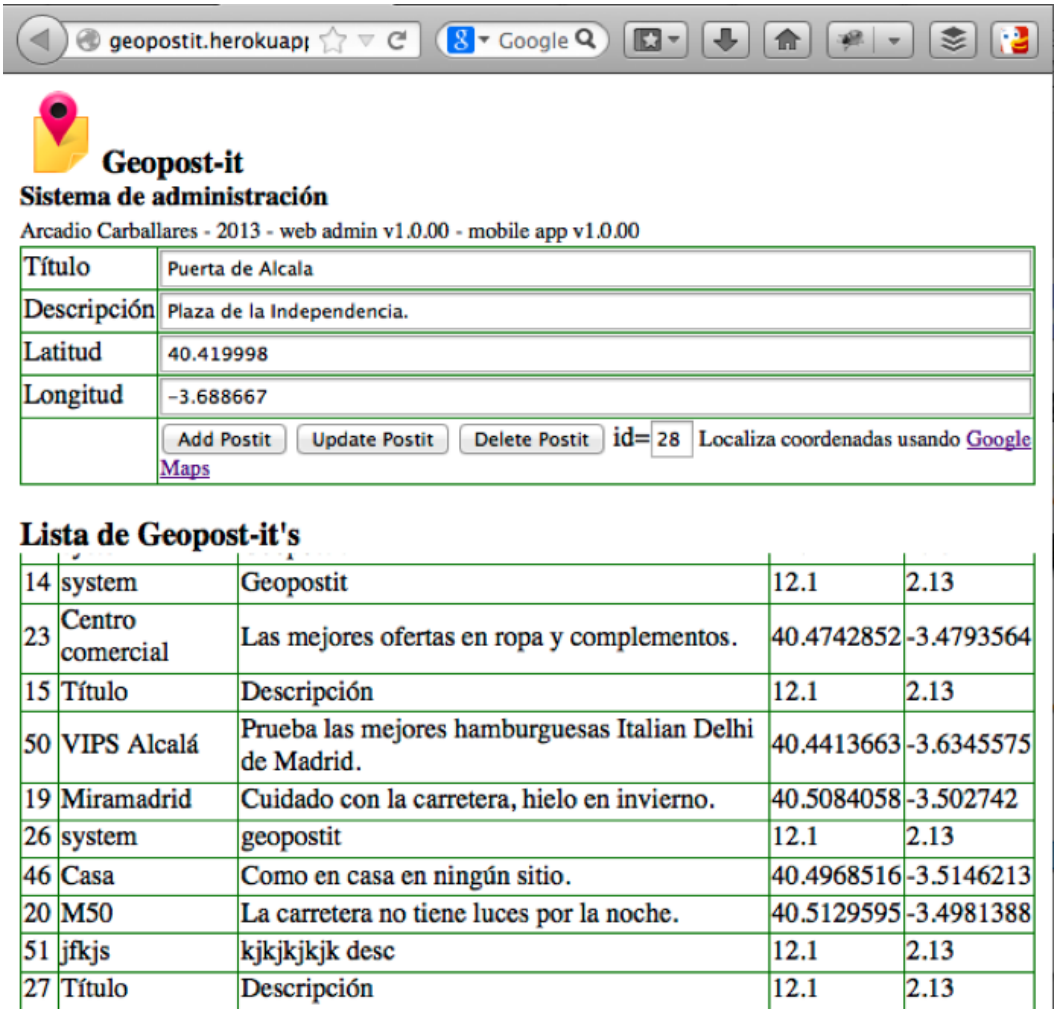
Gracias a este sistema, junto con la **naturaleza Maven** del proyecto, es posible realizar de forma rápida y sencilla los despliegues de las versiones del back-end con la filosofía de heroku de "**despliegue rápido**".

6.5.3. Administrador web

El sistema de back-office ofrece una herramienta para las tareas básicas de gestión y administración de los datos de la aplicación:

- Listado y gestión de post-it del sistema
- Creación de pruebas
- Monitorización de dispositivos

Esta herramienta está realizada en **HTML y AJAX con JQuery** y utiliza los mismos servicios que la aplicación cliente mediante el uso y tratamiento de datos en formato JSON.



6.6. GCM

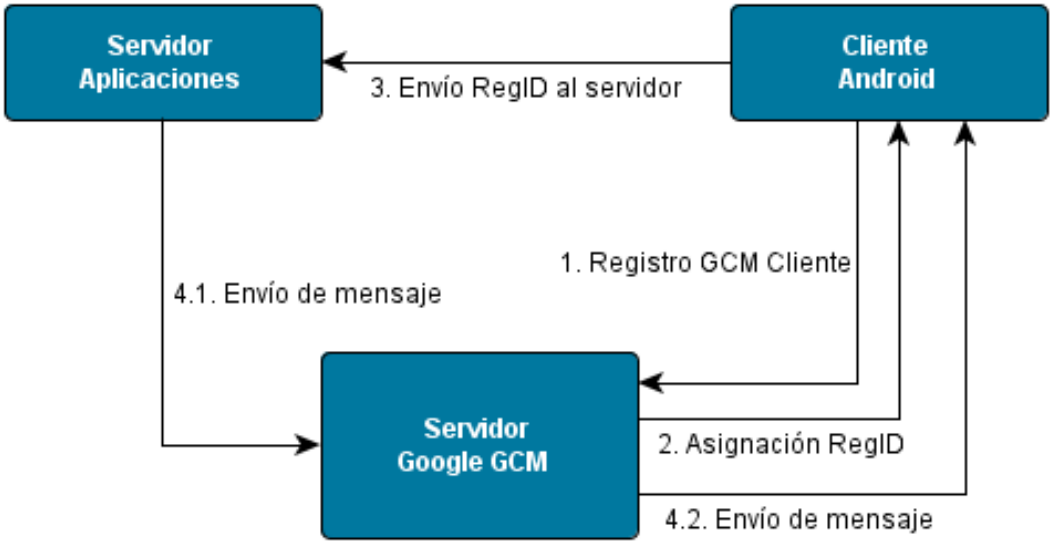
Google Cloud Messaging es un servicio de Google que proporciona las características principales de un sistema de mensajería en la nube que permite que los desarrolladores envíen datos de los servidores a sus aplicaciones de Android.

Puede tratarse de un mensaje corto que indique a la aplicación de Android que hay datos nuevos que se pueden recuperar del servidor (por ejemplo, una película subida por un amigo) o puede ser un mensaje que contenga hasta 4 KB de datos de carga (de esta forma, aplicaciones como los mensajes instantáneos pueden consumir el mensaje directamente).

GCM permite disponer de una **infraestructura gratuita y escalable** para el intercambio de mensajes entre dispositivos Android y servidores intermedios.

El **esquema básico** para el envío y recepción de los mensajes a través de GCM implica:

- Registro del dispositivo en los servidores GCM.
- Envío del mensaje desde cliente o desde el servidor a los servidores de Google.
- Recepción y procesamiento de los mensajes encolados por los clientes de las aplicaciones Android con cuenta en Google Play.

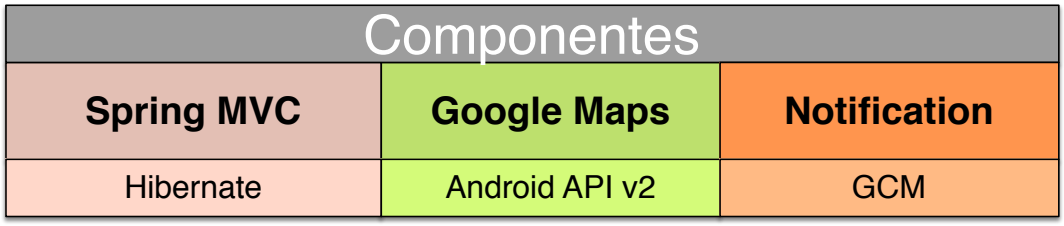


Geopost-it utiliza el sistema GCM HTTP connection server basado en protocolo HTTP para el envío y recepción de mensajes, típicamente sobre formato JSON. Esta variante de GCM está implementada de la siguiente manera:

- En la **parte cliente** la aplicación (con Google Services activo y los permisos necesarios correctamente instalados) rescata del servidor de Google el identificador (normalmente sin variación) y realiza las operaciones típicas de creación y actualización que serán las que desencadenen el mensaje a los sistemas GCM a través de la parte servidora.
- En la **parte servidora** (back-end) se procesan los envíos con identificador de dispositivo por parte de la aplicación Android y se almacenan en la base de datos. Para cada nuevo mensaje o actualización el sistema identifica el proyecto GCM en Google Service y envía un mensaje con la lista de dispositivos identificados en el radio de acción de la localización recibida a los servidores de Google. El sistema GCM se encarga de encolar los mensajes para su consumo por cada dispositivo de la lista de destinatarios.

6.7. Componentes

La arquitectura está formada con una base de componentes que proporcionan las funcionalidades necesarias para cada tipo de operación de manera que en conjunto son capaces de ofrecer una solución completa a los requisitos de la aplicación.

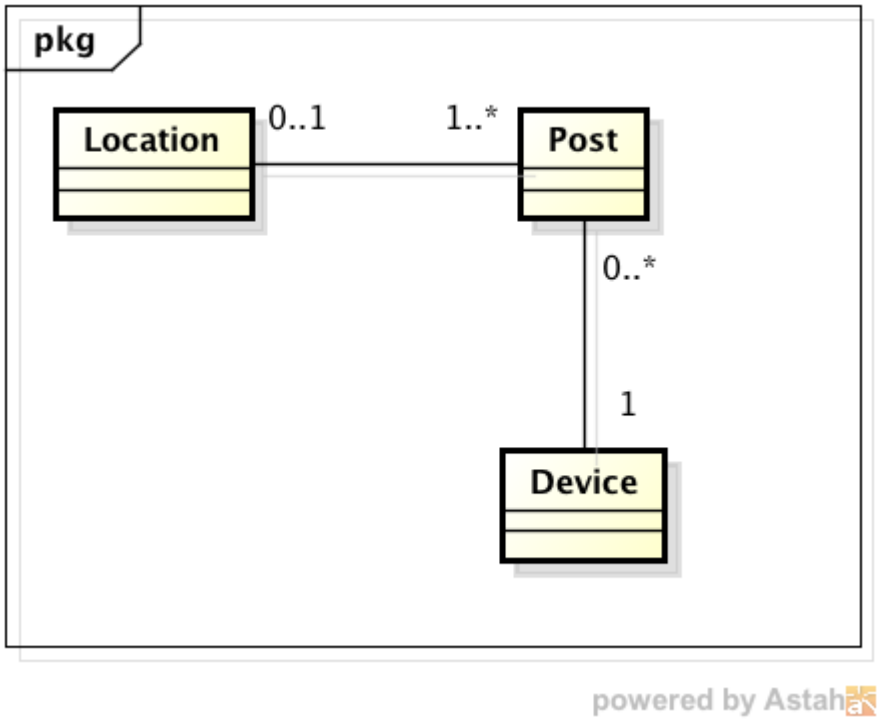


- API Google Maps (https://developers.google.com/maps/?hl=es)
  Google Maps es el conocido servidor de aplicaciones de mapas de Google que ofrece capacidades básicas de GIS para aplicaciones web y aplicaciones móviles. A través de su API los desarrolladores pueden añadir fácilmente funcionalidades a los mapas y programar aplicaciones con características geo-referenciadas:
  - Creación de aplicaciones basadas en la posición
  - Visualización de datos geoespaciales
  - Personalización de mapas
- Google Cloud Messaging for Android
- REST API para Android
- Spring MVC

6.8. Modelo de datos

El modelo de dominio incluye las clases:

- **Device**: objeto que identifica el dispositivo. Se compone de unos datos básicos y un identificador único asociado al dispositivo que se utilizará para asegurar las credenciales unequívocas.
- **Location**: objeto que representa un lugar físico con datos de geo-posicionamiento GIS.
- **Post**: objeto que almacena los datos textuales de la marca y tiene asociada una localización única y un usuario único que la identifica inequívocamente en el sistema back-end.



6.9. Servicios móvil

Las operaciones desde el dispositivo se resumen en invocaciones al controlador del back-end vía http con intercambio de información basada en JSON:

- **Controlador del dispositivo (DeviceController.java)**
  - public List<Device> getItems()
  - public boolean addItem(@RequestBody Device item)
  - public boolean updateItem(@RequestBody Device item)
  - public boolean updateItemByPOST(@RequestBody Device item)
  - public boolean deleteItem(@PathVariable("id") Integer id)

- **Controlador para GCM (GCMController.java)**
  - public boolean addItem(@RequestBody Message message)

- **Controlador para la localización**
  - public List<Location> getItems()
  - public String addItem(@RequestBody Location item)
  - public String updateItem(@RequestBody Location item)
  - public String deleteItem(@PathVariable("id") Integer id)

- **Controlador para la gestión del Postit**
  - public List<Postit> getItems(@RequestParam (required = false) Double latitude, @RequestParam (required = false) Double longitude)

- public boolean addItem(@RequestBody Postit item)
- public boolean updateItem(@RequestBody Postit item)
- public boolean updateItemByPOST(@RequestBody Postit item)
- public boolean deleteItem(@PathVariable("postitId") Integer postitId, @RequestParam String deviceId)
- public boolean deleteItemByGET(@PathVariable("postitId") Integer postitId, @RequestParam String deviceId)

6.10. Servicios back-end

Los servicios expuestos por Spring MVC interactúan con el modelo a través del EntityManager de JPA para las operaciones con la base de datos PostgreSQL alojada en los servicios Cloud de Heroku:

- **Servicios para dispositivo**
  - public boolean addItem (Device item);
  - public boolean updateItem (Device item);
  - public List<Device> getItems();
  - public Device getByPostit(Integer id);
  - public boolean deleteItem (Integer id);
  - public List<String> getDeviceGCMid();

- **Servicios GCM**
  - public void registerDevice (String registerId, Date registerDate);
  - public List<String> getGCMList (Location location, Date registerDate);

- **Servicios de localización**
  - public void addItem (Location item);
  - public void updateItem (Location item);
  - public List<Location> getItems();
  - public void deleteItem (Integer id);

- **Servicios para la gestión del Postit**
  - public boolean addItem (Postit item);
  - public boolean updateItem (Postit item);
  - public List<Postit> getItems();
  - public List<Postit> getItemsByLocation(double latitude, double longitude);
  - public boolean deleteItem (Integer postitId, String deviceId);

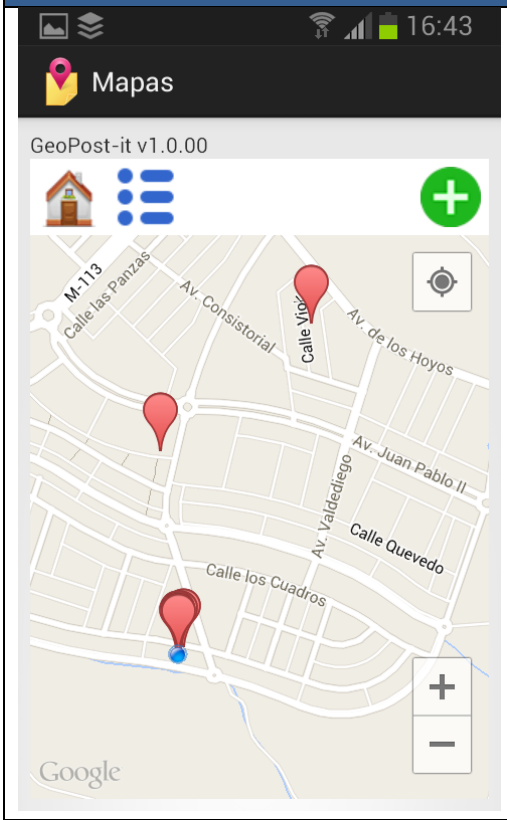
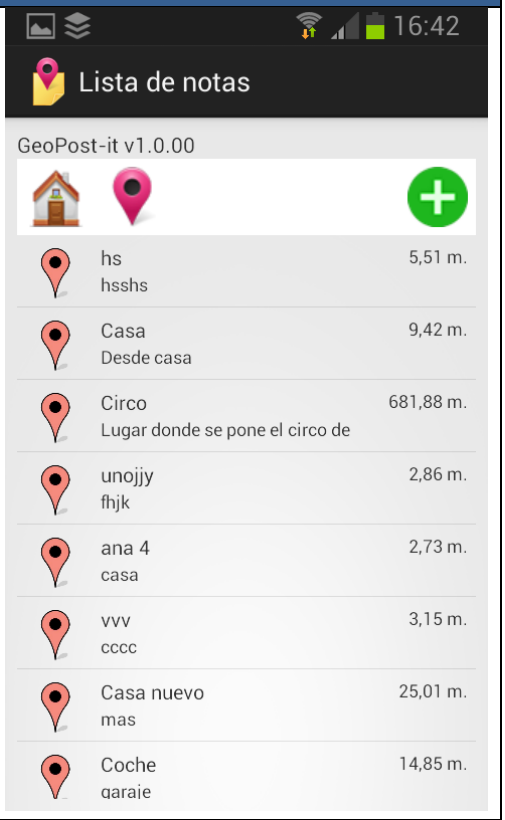
7. Interfaz

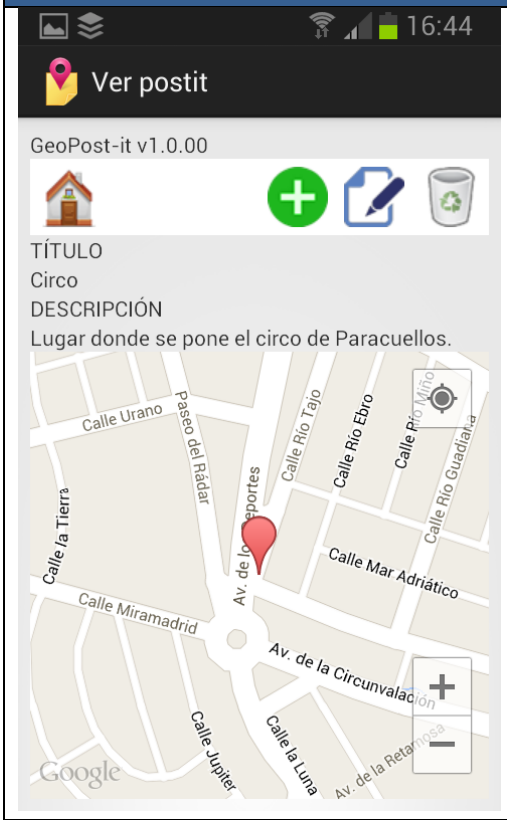
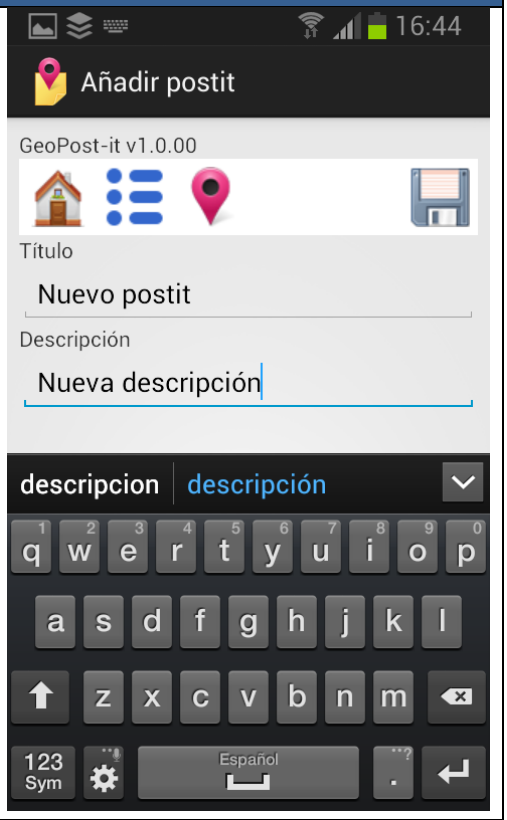
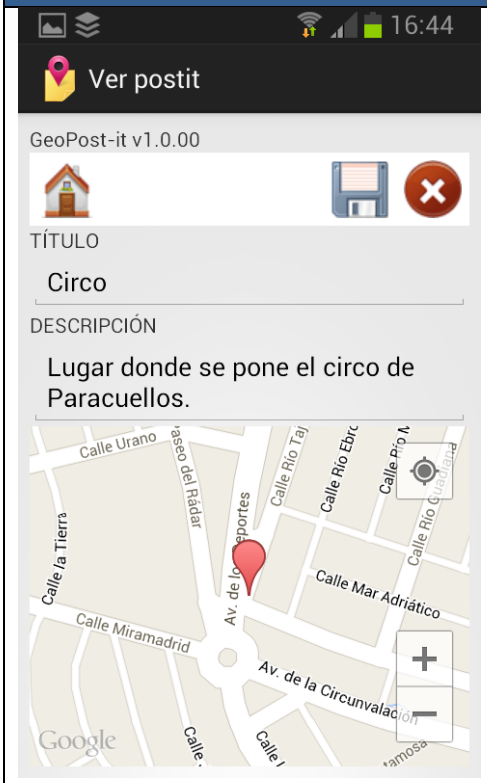
7.1. Interfaz de usuario

La aplicación dispone de una **navegación e interfaz sencillas** tanto para la lectura como para la introducción de datos.
El objetivo es **facilitar al máximo la interacción** con la aplicación para cada una de las operaciones disponibles.

7.2. Pantallas

Principal	Notificación
	
La pantalla inicial con los botones de navegación para acceder a las diferentes secciones de la aplicación: • Información, para ir a la pantalla de información. • Lista, para ir a la pantalla de visualización de marcas en modo lista. • Mapa, para ir a la pantalla de visualización en modo mapa con las marcas geo-posicionadas. • Añadir, para ir a la pantalla de creación de nuevas marcas.	Las notificaciones se muestran como un mensaje de alerta sobre la aplicación activa de la pantalla. El mensaje desaparece tras unos segundos de exposición. Las notificaciones se basan en el sistema GCM de Google y son capaces de interaccionar con cualquier dispositivo basado en Android. Estos mensajes son independientes de la aplicación y se encolan para cada usuario a la espera de que se conecte. Una vez despachados se eliminan de las colas. Los mensajes se entregan con independencia de si la aplicación está activa o no y se muestran en la pantalla del dispositivo en el momento de recibirse.

Posición de marcas en el mapa	Listado de marcas
	
El mapa muestra las marcas próximas geo-posicionadas. Cada marca tiene asociado una acción para dirigir a su ventana de detalle. El usuario selecciona en primera instancia la marca deseada para obtener el globo informativo y luego, si desea ampliar la información, sólo tiene que pulsar sobre la viñeta para que la aplicación redirija a la ventana de Vista de Postit.	La pantalla muestra la lista de marcas en modo listado: • Sólo se muestran las marcas dentro del radio de acción calculado según la posición del dispositivo. • Muestran información básica de la marca y su distancia actual al dispositivo. • Al pulsar en cada una de las marcas la aplicación redirige a la ventana de Vista de Postit.

Vista de marca	Creación de marca
	
Pantalla que muestra una marca seleccionada previamente y donde se muestra su información asociada así como su localización en el mapa.	La pantalla de creación (y de edición) de cada nota muestra los componentes para introducir (o editar) el texto y añadirlo a la ubicación seleccionada.
Edición de marca	**Información**
	
Pantalla de edición marca seleccionada.	Pantalla de información.

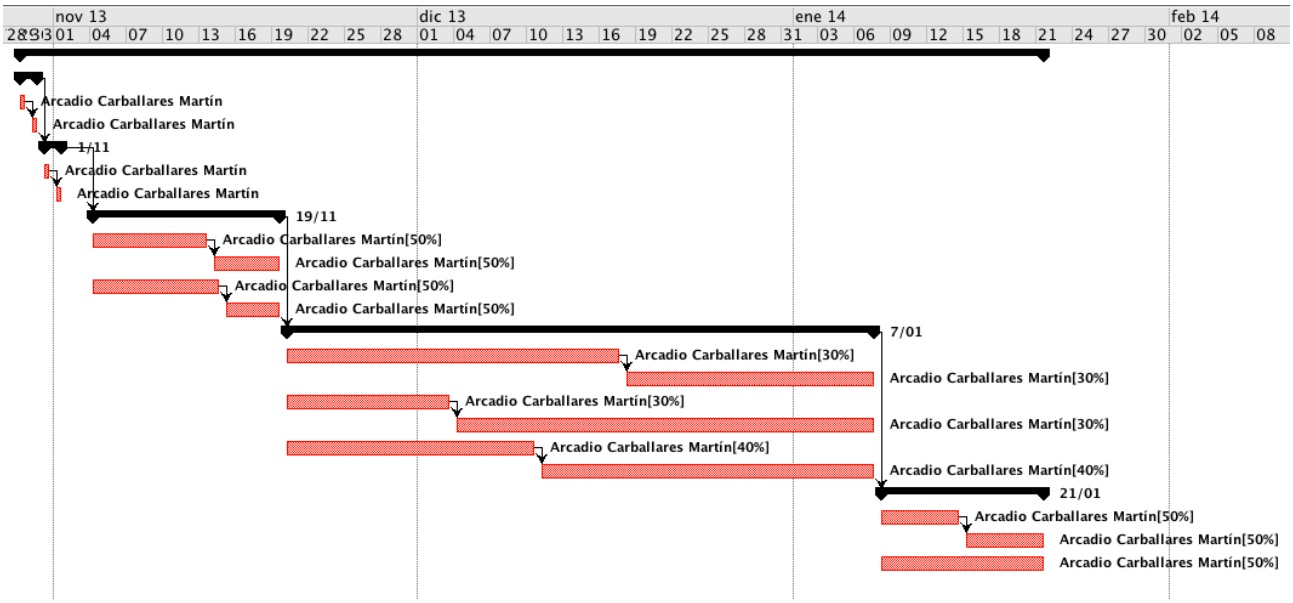
8. Planificación

La planificación refleja la distribución de tareas siguiente:

- Análisis de requisitos
- Análisis Funcional
- Pruebas de concepto
- Diseño Técnico
- Entrega

La programación del desarrollo está definido de forma lineal basado en una metodología de un **ciclo de vida clásico o en cascada** pero está sujeto a una evolución que se aproxima al método utilizado por las **metodologías ágiles,** donde se solapan tareas no dependientes para avanzar en paralelo y permitir la generación de artefactos funcionales.

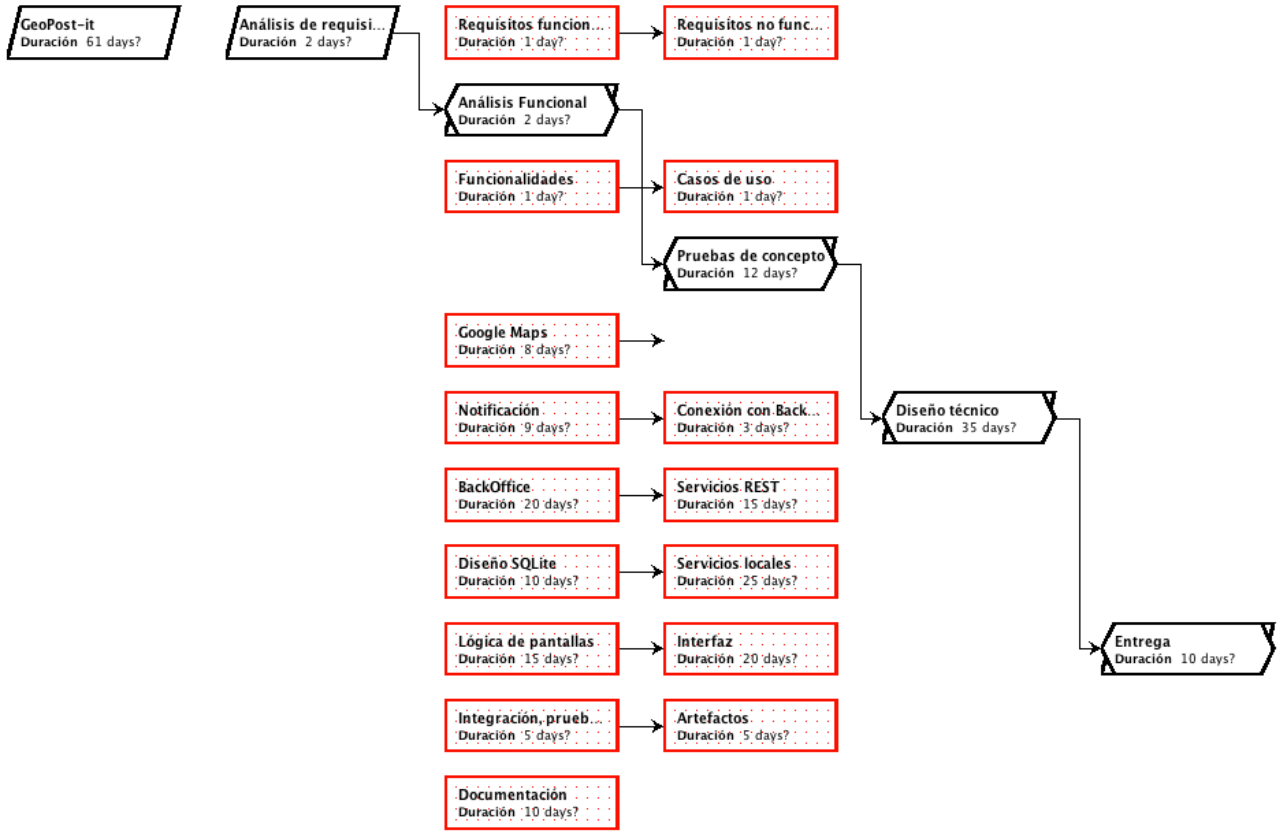
8.1. Diagrama de Gantt



8.2. Calendario de tareas

	Nombre	Duración	Inicio	Terminado	Predecesores	Nombres del Recurso
1	**GeoPost-it**	61 days?	**29/10/13 8:00**	**21/01/14 17:00**		
2	**Análisis de requisitos**	2 days?	**29/10/13 8:00**	**30/10/13 17:00**		
3	Requisitos funcionales	1 day?	29/10/13 8:00	29/10/13 17:00		Arcadio Carballares Martín
4	Requisitos no funcioanles	1 day?	30/10/13 8:00	30/10/13 17:00	3	Arcadio Carballares Martín
5	**Análisis Funcional**	2 days?	**31/10/13 8:00**	**1/11/13 17:00**	**2**	
6	Funcionalidades	1 day?	31/10/13 8:00	31/10/13 17:00		Arcadio Carballares Martín
7	Casos de uso	1 day?	1/11/13 8:00	1/11/13 17:00	6	Arcadio Carballares Martín
8	**Pruebas de concepto**	12 days?	**4/11/13 8:00**	**19/11/13 17:00**	**5**	
9	Google Maps	8 days?	4/11/13 8:00	13/11/13 17:00		Arcadio Carballares Martín[50%]
10	Posicionar marca	4 days?	14/11/13 8:00	19/11/13 17:00	9	Arcadio Carballares Martín[50%]
11	Notificación	9 days?	4/11/13 8:00	14/11/13 17:00		Arcadio Carballares Martín[50%]
12	Conexión con BackOffice	3 days?	15/11/13 8:00	19/11/13 17:00	11	Arcadio Carballares Martín[50%]
13	**Diseño técnico**	35 days?	**20/11/13 8:00**	**7/01/14 17:00**	**8**	
14	BackOffice	20 days?	20/11/13 8:00	17/12/13 17:00		Arcadio Carballares Martín[30%]
15	Servicios REST	15 days?	18/12/13 8:00	7/01/14 17:00	14	Arcadio Carballares Martín[30%]
16	Diseño SQLite	10 days?	20/11/13 8:00	3/12/13 17:00		Arcadio Carballares Martín[30%]
17	Servicios locales	25 days?	4/12/13 8:00	7/01/14 17:00	16	Arcadio Carballares Martín[30%]
18	Lógica de pantallas	15 days?	20/11/13 8:00	10/12/13 17:00		Arcadio Carballares Martín[40%]
19	Interfaz	20 days?	11/12/13 8:00	7/01/14 17:00	18	Arcadio Carballares Martín[40%]
20	**Entrega**	10 days?	**8/01/14 8:00**	**21/01/14 17:00**	**13**	
21	Integración, pruebas y optir	5 days?	8/01/14 8:00	14/01/14 17:00		Arcadio Carballares Martín[50%]
22	Artefactos	5 days?	15/01/14 8:00	21/01/14 17:00	21	Arcadio Carballares Martín[50%]
23	Documentación	10 days?	8/01/14 8:00	21/01/14 17:00		Arcadio Carballares Martín[50%]

8.3. Distribución de tareas



8.4. Carga de trabajo real final

La **carga de trabajo final en lo que se refiere estrictamente al desarrollo** (sin contar con los análisis previos al TFC) con imputaciones reales hasta la versión final de la aplicación se distribuyó de la siguiente manera:

• Documentación	12:30h.
• Análisis de requerimientos	02:30h.
• Análisis funcional	02:30h.
• Arquitectura	03:00h.
• Entregables	04:00h.
• Gestión de post-it	11:00h.
• Integración, optimización y pruebas	02:00h.
• Interfaz	18:00h.
• Mapas	07:00h.
• Pruebas de concepto	06:30h.
• GCM-Push Notification	09:00h.
• Servicios remotos	18:00h.
TOTAL	**100:00h.**

Coeficiente de reducción

La estimación de jornadas (normalmente equivalente a 8h.) tiene que reducirse al tiempo estimado de dedicación disponible por día. Asumiendo una carga diaria de 3h. de dedicación por jornada se traduce en una reducción del 65%.

Cuadro comparativo

A continuación se detalla un **cuadro comparativo sobre las horas estimadas** (sin coeficiente de reducción) por jornada estipulada en el diagrama de Gantt y las **horas de desarrollo real**, registradas durante el proceso de implementación. Hay que tener en cuenta las horas reales reflejan registros de actividad y no contemplan otro tipo de actividades complementarias que tienen lugar en las fases de un proyecto (reuniones, cambio de estrategias, negociaciones, desviaciones por fuga de recursos, etc.).

Tasa de acierto

Una vez aplicado el ajuste de reducción el balance final entre el tiempo proyectado y el tiempo de dedicación real es de una desviación de 24,52 horas lo que su pone una tasa del +15% aproximadamente.

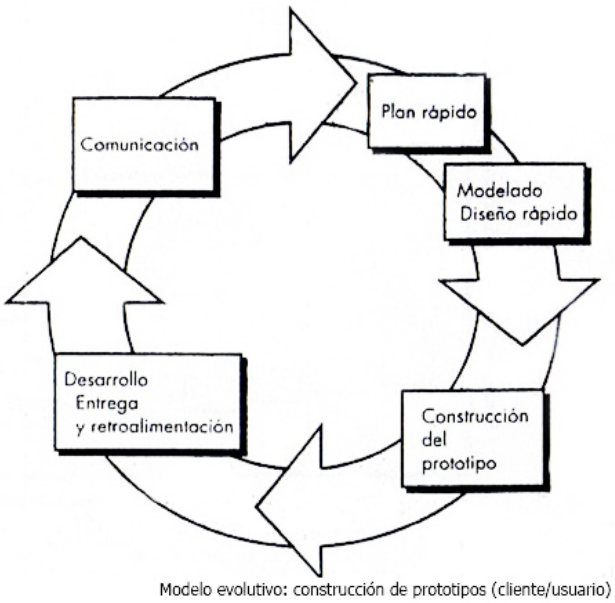
Concepto	Horas Estim.	Horas reales	Desviación
Análisis de requisitos	**16:00h.**	**02:30h.**	La estimación debe completarse con el tiempo dedicado en los análisis previos (+25h.)
- Requisitos funcionales	08:00h.	01:00h.	
- Requisitos no funcionales	08:00h.	01:30h.	
Análisis funcional	**16:00h.**	**02:30h.**	La estimación debe completarse con el tiempo dedicado en los análisis previos (+25h.)
- Funcionalidades	08:00h.	01:00h.	
- Casos de uso	08:00h.	01:30h.	
Pruebas de concepto	**96:00h.**	**22:30h.**	La estimación debe completarse con el tiempo dedicado en los análisis previos (+45h.)
- Google Maps	32:00h.	07:00h.	
- Posicionar marca	16:00h.	01:00h.	
- Notificación	30:00h.	09:00h.	
- Conexión con Backoffice	18:00h.	05:30h.	
Diseño técnico	**280:00h.**	**50:00h.**	La estimación refleja el ahorro de tiempo que permite el uso de los frameworks y arquitecturas elegidas para el proyecto. La parte de SQLite se desestimó para el proyecto.
- Backoffice	50:00h.	03:00h.	
- Serviciso REST	40:00h.	18:00h.	
- Diseño SQLite	40:00h.	-	
- Servicios locales	60:00h.	11:00h.	
- Lógica de pantallas	40:00h.	05:00h.	
- Interfaz	50:00h.	18:00h.	
Entrega	**80:00h.**	**43:00h.**	
- Integración, pruebas y optimización	20:00h.	02:00h.	La generación de artefactos y tareas de integración se benefician de la reducción de tiempo por el uso de una infraestructura en Cloud.
- Artefactos	20:00h.	04:00h.	
- Documentación	40:00h.	12:30h.	Hay que considerar los trabajos previos de la documentación en las fases previas al desarrollo.
TOTAL Desarrollo	488:00h	100:00h.	
Coeficiente reducción	**65%**	**0.0**	
TOTAL con reducción	170:48h.	100:00h.	
Trabajos previos al desarrollo		**95:00h.**	
- Análisis previos PEC 1	-	25:00h.	
- Análisis previos PEC 2	-	25:00h.	
- Pruebas	-	45:00h.	
TOTAL	170:48h.	195:00h.	

Estas tareas no incluyen **tareas previas de documentación y pruebas** sobre las tecnologías que se usan y, en otros casos, incluyen **conocimientos previos y metodologías** que han ayudado a agilizar los tiempos así como el uso de herramientas, IDE's y frameworks orientados a la mejora de la productividad.

9. Mantenimiento

9.1. Actualizaciones y Correcciones

Las actualizaciones y las correcciones se realizan sobre un **modelo de desarrollo basado en versiones** (o prototipos) funcionales y en cada una de ellas se implementan las nuevas funcionalidades o cambios requeridos por el cliente y en los que se dan solución a los bugs del sistema.



Modelo evolutivo: construcción de prototipos (cliente/usuario)

Para esta aplicación se propone la entrega de **3 versiones en fase Beta** con funcionalidad incremental y correcciones de bugs y una **entrega Release final** con la implementación de los requisitos analizados en las fases previas.

9.2. Mejoras propuestas

- Posicionamiento de **ficheros** y **material audiovisual**.
- **Localización** para varios idiomas.
- **Brújula** de navegación para orientar al usuario hacia una marca.
- **Plataforma** para compartir las notas con otros usuarios (notas públicas, privadas y grupales).
- **Categorización** de los post-it para que se puedan recibir en función de los temas de interés del usuario.
- **Limitar las notificaciones** a los dispositivos que están dentro del radio de acción.
- **Filtrado** de post-it en función de categoría, actividad, distancia y otros parámetros para que el usuario acote la visualización a sus intereses.
- Configuración del **radio de acción** de los post-it para que cada dispositivo pueda modificar el alcance de las marcas a visualizar.
- Implementar **acceso restringido** para el backoffice.
- Generar una **API doc** para las clases del proyecto.

10. Entregables

10.1. Prototipo

Las **entregas parciales** se componen de:

- Beta 1:
  - Primera aproximación a la aplicación
  - Diseño wireframe del interfaz
  - Implementación de las primeras funcionalidades tras la etapa del prototipado y de las pruebas de concepto.
  - Prototipo del panel de administración para la gestión de pruebas.
- Beta 2:
  - Todas las funcionalidades implementadas
  - Solución a los bugs reportados de la beta 1.
  - Versión final del panel de administración del back-end y solución a los problemas de integración con la beta 1.
- Beta 3:
  - Acabado final del interfaz con el aspecto final y las pantallas definitivas.
  - Resolución de bugs de las etapas anteriores.
  - Pantalla de información y créditos.

La **entrega final** se compone de una **versión operativa de la aplicación** como fichero instalable Android (.apk).

10.2. Documentación

La documentación del proyecto consta de la **memoria final** y el **código fuente** tanto del back-end como de la aplicación Android.

Además de las referencias para los desarrolladores de los componentes utilizados:

- Google Maps (https://developers.google.com/maps/?hl=es)
- GCM (http://developer.android.com/google/gcm/index.html)
- Android developers (http://developer.android.com/index.html)
- Hibernate (http://hibernate.org/)
- Spring MVC (http://spring.io/)
- Heroku (http://heroku.com/)

11. Conclusiones

GeoPost-it es un **acercamiento a las tecnologías para plataformas móviles** y a las posibilidades de movilidad que ofrecen los dispositivos informáticos actuales. Pero no sólo es una propuesta móvil, sino que es un **conjunto de soluciones técnicas e infraestructuras** en las que intervienen varias de las tecnologías y metodologías más novedosas del mercado. En el sentido técnico, GeoPost-it representa un sistema general desde la parte cliente hasta la parte servidora en donde intervienen diferentes etapas que completan un escenario complejo para dar cobertura a los servicios propuestos.

11.1. Valoración de la tecnología empleada

El desarrollo de aplicaciones basadas en tecnología Android ofrece las ventajas propias de las aplicaciones nativas, lo que permite un mayor control sobre las funciones físicas del dispositivo (cámaras, GPS, galerías, etc.) pero, además, para disponer de un servicio completo en el que se promueva una comunidad o una gestión de datos, es necesario implementar una aplicación back-end de apoyo a la versión *mobile*. Esta variedad de funcionalidades obliga a contemplar en el desarrollo **diferentes tecnologías cliente y servidor** que en conjunto ofrezcan la solución que se busca.

En general, según la experiencia de este proyecto, las **ventajas y desventajas de las tecnologías usadas** se pueden resumir en el siguiente cuadro:

Tecnología	Ventajas	Desventajas
Android	• Programación en un lenguaje de amplia difusión como JAVA. • API's nativas para el acceso a los recursos de los dispositivos. • Amplio soporte y comunidad de desarrolladores. • Amplia documentación y tutoriales oficiales. • Se integra muy bien con otros sistemas proporcionados por Google (Maps, GCM)	• El sistema de pruebas y máquinas virtuales es excesivamente lento. • El lenguaje no está tan logrado como JAVA y el IDE es lento y pesado. • El desarrollo de la interfaz es tedioso y poco logrado.
Spring MVC con Heroku	• Infraestructura en Cloud que libera recursos y configuraciones al desarrollador. • Framework Spring muy extendido y probado. Fácil de integrar con este tipo de soluciones. • Respuesta de conexión aceptable y escalable (previo pago). • Control de software por Git con despliegue rápido.	• Configuraciones complicadas para temas como seguridad y bases de datos. • Limitaciones propias de su infraestructura.
GCM	• API de desarrollo propia. • Muy buena integración con	• Limitaciones a escalas grandes.

	Android.	
Google Maps	• API de desarrollo propia. • Muy buena integración con Android.	• La implementación de las personalizaciones es un poco tediosa.

11.2. Valoración de la metodología empleada

La **metodología ágil** (similar a SCRUM) que se ha utilizado a lo largo del desarrollo ha resultado adecuada para la generación de prototipos con iteraciones complementarias sobre una base de aplicación funcional.

Esta metodología **ha reducido el impacto de una versión final** con errores o funcionalidades no desarrolladas según la idea inicial. En todo momento las versiones beta del producto han ido ofreciendo más funcionalidad o corrigiendo fallos previos, de manera que se ha podido tener una visión completa de la aplicación según la fase de análisis antes de su entrega final y disponer de una alta fiabilidad para garantizar su implementación completa correctamente.

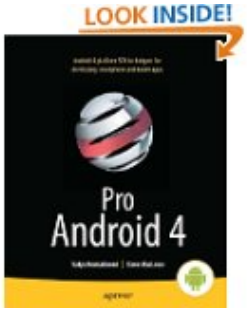
11.3. Valoración de las herramientas empleadas

Las herramientas utilizadas en el desarrollo ha sido las que proporciona (recomienda) Google para el desarrollo de Android en el caso de **ADT** y el uso de **Eclipse Kepler** con los plug-ing de Heroku y STS (Spring Tools Suite) para la integración con la arquitectura Spring MVC en Cloud.
En general la respuesta de Eclipse es excelente dado que es un IDE muy extendido y probado.
En el caso de ADT se aprecia un **consumo de recursos** mayor de lo habitual debido al desarrollo con los entornos de prueba de las AVD (Android Virtual Device) que son bastante lentos e improductivos.

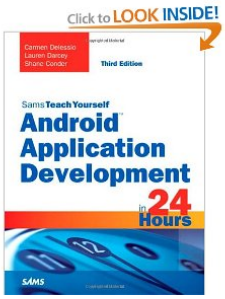
A lo largo de este documento y del desarrollo de la TFC se intenta transmitir la realidad de un entorno en el que las funcionalidades y recursos que proporcionan los móviles representan una **novedosa alternativa a las soluciones del software tradicional** y permiten implementar y desarrollar otros mecanismos evolucionados de interacción. Esta situación obliga a replantear y adaptar los métodos tradicionales de desarrollo y elaborar nuevas metodologías de software adaptadas a estas necesidades. Pero también **representa una oportunidad** para enfocar la programación desde el punto de vista del usuario final.

12. Bibliografía

- **Pro Android 4** by Satya Komatineni and Dave MacLean, Apress Media (2012)



- **Android Application Development in 24 Hours** by Sams Teach Yourself (3rd Edition)



- **Android in Action** by Frank Ableson and Robi Sen (2011)

- **Programming Android: Java Programming for the New Generation of Mobile Devices** by Zigurd Mednieks, Laird Dornin, G. Blake Meike and Masumi Nakamura, O'Reilly (2012)