**PROYECTO FIN DE CARRERA**

| | |
|---|---|
| **Título:** | "Diseño de una Aplicación Social en Android con Localización basada en Realidad Aumentada" |
| **Autor:** | Eduardo González Martín |
| **Tutor:** | Carlos Ángel Iglesias Fernández |
| **Departamento:** | Ingeniería de Sistemas Telemáticos |

**MIEMBROS DEL TRIBUNAL CALIFICADOR**

| | |
|---|---|
| **Presidente:** | D. Gregorio Fernández Fernández |
| **Vocal:** | D. Mercedes Garijo Ayestarán |
| **Secretario:** | D. Carlos Ángel Iglesias Fernández |
| **Suplente:** | D. Marifeli Sedano Ruiz |

**Fecha de Lectura:**

**Calificación:**

**UNIVERSIDAD POLITÉCNICA DE MADRID**

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos Grupo de Sistemas Inteligentes


PROYECTO FIN DE CARRERA

# DISEÑO DE UNA APLICACIÓN SOCIAL EN ANDROID CON LOCALIZACIÓN BASADA EN REALIDAD AUMENTADA

Eduardo González Martín

Julio de 2013

*A mis padres, por haberse privado de tantas cosas para poder darme una oportunidad en la vida.*

# Resumen

Esta memoria presenta el resultado del desarrollo de la aplicación para Android "Contacta!", ganadora del segundo premio del "II Concurso de Desarrollo de Aplicaciones para Android", promovido por Telefónica y la Cátedra Telefónica Aplicaciones y Servicios Móviles en la Universidad Politécnica de Madrid. La aplicación consiste en una red social formada por los contactos que contiene la agenda del móvil cuya finalidad radica en poder ver dónde se encuentran en un mapa o mediante realidad aumentada, ver el estado de cada uno, llamar, enviar SMS y chatear con ellos. Una vez instalada la aplicación, ésta detecta automáticamente qué contactos la tienen instalada y ofrece una serie de funciones avanzadas que se pueden realizar con ellos, como establecer alertas de proximidad para saber cuándo se encuentran cerca o las ya mencionadas anteriormente.

**Palabras clave:** Android, Aplicación móvil, Base de Datos, Java, JSON, Realidad Aumentada, Red Social, Servlet, Smartphone.

# Abstract

This Project presents the results of the development of the Android app "Contacta!", which won the second prize in the " II Concurso de Desarrollo de Aplicaciones para Android", sponsored by Telefónica and Cátedra Telefónica Aplicaciones y Servicios Móviles at the Universidad Politécnica de Madrid. The application consists of a social network formed by the contacts stored in the mobile agenda, and its purpose is to be able to see where they are on a map or using augmented reality, see their status, make a call, send SMS and chat with them. After installing the application, it automatically detects which contacts have it installed and offers several advanced features that you can do with them, as set proximity alerts to know when they are near or those already mentioned above.

**Keywords:** Android, Augmented Reality, Database, Java, JSON, Mobile Application, Servlet, Social Network.

# Agradecimientos

Me gustaría dar las gracias a mi familia, sin la cual no podría haber llegado hasta aquí y por haberme dado la oportunidad de estudiar esta carrera tan lejos de casa y por apoyarme incondicionalmente en los momentos más difíciles.

A Anita, por cuidarme y soportarme tantos días mientras desarrollaba mis aplicaciones.

A mis amigos, por haberme hecho más agradable mi estancia en Madrid y por todos los buenos momentos que hemos pasado juntos.

Y por supuesto a Carlos Ángel, por haberme enseñado a programar y ayudarme durante todos estos años.

# Índice de Contenidos

## Índice de Contenidos

# Índice de Tablas

# Índice de Figuras

# Capítulo 1: Introducción

*En este capítulo se hace una introducción a este proyecto de fin de carrera. Se describirán las motivaciones que han llevado a realizarlo, los objetivos establecidos y cómo se ha estructurado la memoria*

## 1.1 Motivaciones

La evolución de Internet desde su origen hasta ahora, tal y como la conocemos, ha pasado por distintas fases marcadas por una característica dominante muy concreta y otras, en segundo plano, que han sido menos relevantes, quizás, pero que han marcado las etapas de la Red.

En los últimos años, destaca sobre todo lo demás la evolución de la tecnología móvil, cuya comunidad de desarrollo se encuentra en un punto de inflexión. Los usuarios móviles demandan más opciones, más oportunidades para personalizar sus teléfonos, y una mayor funcionalidad. Los operadores móviles quieren ofrecer contenido de valor agregado a sus abonados en una forma manejable y lucrativa. Los desarrolladores de móviles quieren la libertad para desarrollar la tremenda demanda de aplicaciones móviles por parte de los usuarios con un mínimo de obstáculos para el éxito. Actualmente, los fabricantes de teléfonos quieren una plataforma estable, segura y asequible para alimentar sus dispositivos. Hasta ahora ninguna plataforma móvil ha sabido responder adecuadamente a las necesidades de todas las partes.

Por este motivo nace Android [1], un sistema operativo principalmente para móviles basado en Linux, desarrollado inicialmente por Android Inc., que posteriormente fue comprado por Google y se convierte en el principal producto de la Open Handset Alliance, un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio.

Android cambia completamente la comunidad de desarrollo móvil. Una plataforma innovadora, abierta y completamente gratuita, que está bien posicionada para hacer frente a las crecientes necesidades del mercado móvil. A finales del año 2012, Android cuenta con el 70% de la cuota de smartphones en todo el mundo, y con una tienda con más de 700.000 aplicaciones, llamada Google Play [2]. Android tiene el potencial para quitar las barreras del éxito en el desarrollo y venta de una nueva generación de software de aplicaciones móviles. Así como las plataformas PC y Macintosh crearon mercados para el software de escritorio y servidor, Android, proporcionando un entorno estándar de aplicaciones de teléfono móvil, crea un mercado y da la oportunidad a los desarrolladores de aplicaciones a beneficiarse.

Teniendo todo esto en cuenta, se ha decidido realizar una aplicación para este sistema, ya que ofrece un gran mercado lleno de posibilidades en el que un desarrollador individual puede dar a conocer fácilmente sus aplicaciones en todo el mundo con un coste muy bajo.

## 1.2 Objetivos

El presente proyecto tiene como objetivo el desarrollo de una aplicación para Android consistente en una red social formada por los contactos que contiene la agenda del móvil. Se podrá ver dónde se encuentran (tanto en el mapa como por realidad aumentada), el estado, llamar, enviar SMS y chatear con ellos.

Este objetivo general se desglosa en los siguientes subobjetivos:

1. Profundizar en el conocimiento de desarrollo de aplicaciones móviles y del diseño de servicios.

2. Revisar el estado del arte en el uso de realidad virtual para localización, con especial énfasis en su aplicación a redes sociales.

3. Evaluar las posibilidades disponibles para el sistema operativo Android.

4. Diseñar el servicio que soporte la red social, diseñando tanto el cliente Android como el servidor de información de la red.

5. Evaluar el diseño mediante el desarrollo de un prototipo.

Se utilizará la versión de Android 2.3.3, con su SDK v10, por ser la más extendida, ya que abarca el 50% de todos los dispositivos con el sistema operativo en cuestión. Para el posicionamiento de los contactos se deberá analizar las posibilidades de realidad aumentada para mejorar las relaciones sociales en aplicaciones móviles, estudiando los posibles motores de realidad aumentada que pueda haber.

A su vez, también será necesaria la creación de un servidor que gestione y albergue todos los datos de los usuarios y permita la interacción entre ellos, por lo que se evaluarán las tecnologías que mejor cumplan esta función.

Para poder comprobar la viabilidad del proyecto, será necesario desarrollar un prototipo.

## 1.3 Estructura de la memoria

Para dar a conocer el contenido de cada capítulo, en esta sección se va a introducir cada uno de ellos con una breve explicación de su contenido. El documento completo consta de seis capítulos:

- Capítulo 1: consiste en una introducción al proyecto fin de carrera para dar a conocer una primera visión del proyecto, cuáles son los objetivos y cómo está estructurado.

- Capítulo 2: en este capítulo se muestran las tecnologías necesarias que han hecho posible la realización de este proyecto.

- Capítulo 3: realiza un análisis de funcional del proyecto que muestra los requisitos definidos y las acciones que podrán realizar los usuarios con el sistema desarrollado.

- Capítulo 4: este capítulo presenta la arquitectura completa del sistema completo. Al principio se describen todos los elementos y finalmente se realiza en análisis en profundidad.

- Capítulo 5: describe un ejemplo real de funcionamiento, en el que se puede observar qué puede hacer un usuario con el desarrollo hecho en este proyecto.

- Capítulo 6: por último, se realiza un análisis de las conclusiones a las que se ha llegado tras realizar este proyecto y los posibles trabajos futuros a realizar para mejorarlo.

# Capítulo 2: Tecnologías habilitadoras

*En este capítulo se describen las tecnologías clave que se han empleado en la elaboración del proyecto. En primer lugar, se introducirá la plataforma de desarrollo Java Enterprise Edition, que se ha utilizado para desarrollar el sistema, y posteriormente se seguirá con la descripción de las tecnologías necesarias para lograr realizar la parte de servidor y cliente.*

## 2.1 Interacción entre tecnologías

En el siguiente diagrama se pueden observar las dependencias entre todas las tecnologías utilizadas:

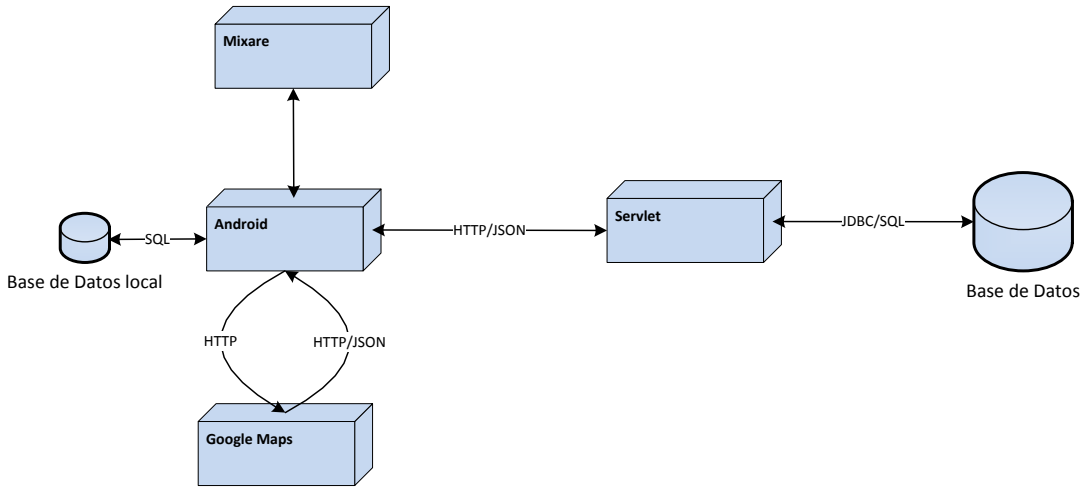


Figura 2.1: Diagrama de interacción de tecnologías

Se puede observar que hay dependencia entre las diferentes tecnologías utilizadas.

- Para poder introducir y obtener la información de los usuarios, se precisan de **bases de datos**, las cuales se acceden mediante el lenguaje *SQL*.
- La aplicación solicita la información que necesita mediante HTTP y se le es devuelta mediante JSON.
- El lenguaje Java del servidor (**Servlets**) permite llevar a cabo la comunicación entre la aplicación y la base de datos tanto para introducir como para recibir resultados, haciendo la función de intermediario entre ambos.
- Todas las tecnologías conjuntas forman la aplicación “Contacta!”.

## 2.2 Java Enterprise Edition

JavaEE [3] es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en lenguaje de programación Java y que se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. La especificación original se denominó J2EE y fue desarrollada por Sun Microsystems.

Las razones que empujan a la creación de la plataforma JEE fueron:

- **La programación eficiente:** para conseguir productividad es importante que los equipos de desarrollo tengan una forma estándar de construir múltiples aplicaciones

en diversas capas (cliente, servidor web, etc.) En cada capa se necesitarán diferentes herramientas:

- **Cliente:** aplicaciones, Java, Applets.
- **Servidor:** Servlets, JSP.

Con JEE tenemos una tecnología estándar, un único modelo de aplicaciones.

- **Integración:** los equipos de ingeniería precisan estándares que favorezcan la integración entre diversas capas de software.
- **Extensibilidad frente a la demanda del negocio:** en un contexto de crecimiento de números de usuarios es precisa la gestión de recursos, como conexiones a BBDD y transacciones.

La arquitectura JEE implica un modelo de aplicaciones distribuidas en diversas capas o niveles:

- La capa cliente admite diversos tipos de clientes (HTML, Applet, aplicaciones Java, etc.).
- La capa intermedia contiene subcapas (el contenedor web y el contenedor EJB).
- La tercera capa dentro de esta visión sintética es la de aplicaciones 'backend' como ERP, EIS, bases de datos, etc.

Como se puede ver un concepto clave de la arquitectura es el de contenedor, que dicho de forma genérica no es más que un entorno de ejecución estandarizado que ofrece unos servicios por medio de componentes.

Un tipo de contenedor es el contenedor web, maneja la ejecución de los Servlets y páginas JSP. Estos componentes se ejecutan sobre un servidor de aplicaciones.

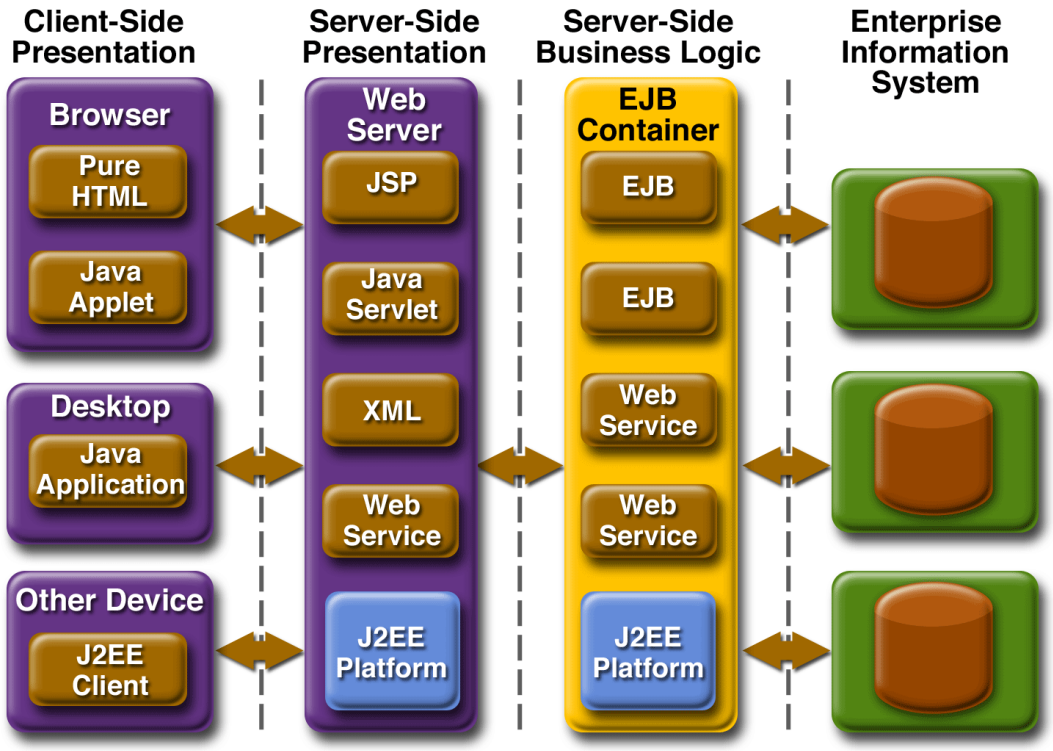


Figura 2.2: Arquitectura de JEE

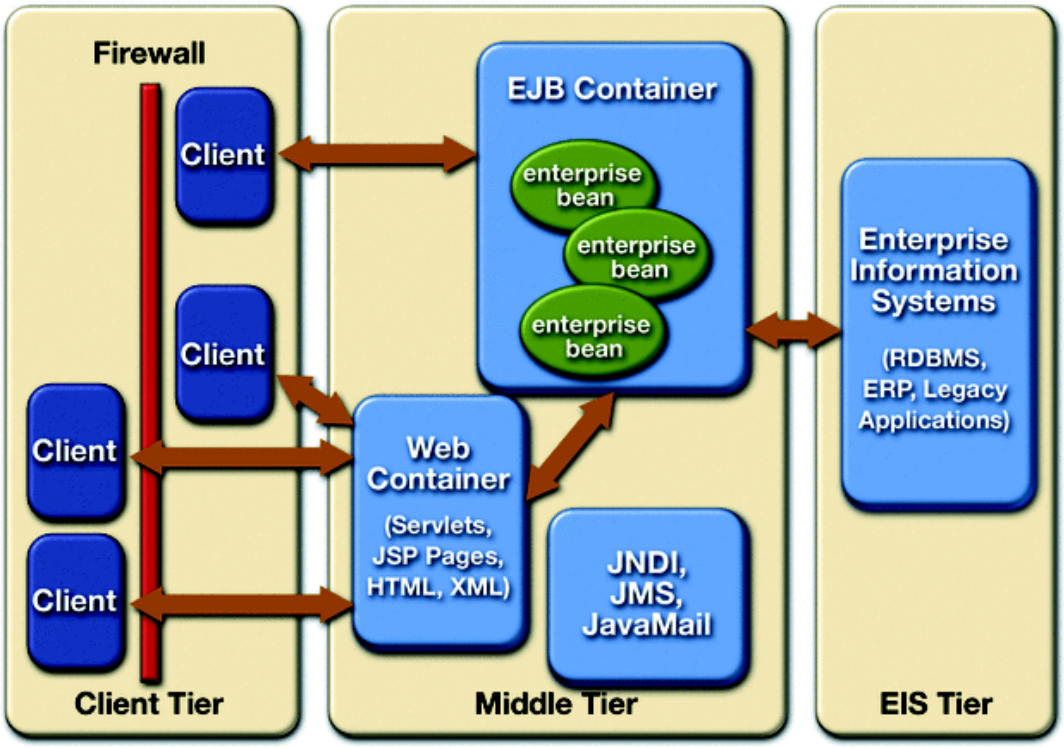


Figura 2.3: Modelo de capas de JEE

Un servidor de aplicaciones es un software de infraestructura que proporciona una serie de servicios a aplicaciones que corren en su interior. El servidor de aplicaciones va a ser quien ejecute y controle la ejecución de nuestras aplicaciones web. El servidor de aplicaciones será quien realmente reciba la petición http. Él analizará dicha petición y almacenará toda la información relativa a ella en objetos Java.

La arquitectura de un servidor de aplicaciones incluye una serie de subsistemas:

- **Servidor HTTP**: también denominado servidor Web o servidor de páginas. Un ejemplo, el servidor *Apache*.
- **Contenedor de aplicaciones** o contenedor *Servlet*. Un ejemplo, *Tomcat*.

Los contenedores incluyen descriptores de despliegue, que son archivos XML que nos sirven para configurar el entorno de ejecución: rutas de acceso a aplicaciones, control de transacciones, parámetros de inicialización, etc.

La **función** del servidor en este proyecto será la de hacer de mediador entre la aplicación cliente y la base de datos, de manera que atenderá las peticiones que le lleguen desde el cliente, las procesará y responderá buscando información en dicha base de datos.

## 2.3 Servlets

Un *Servlet* [4] es un programa ejecutado en el servidor. Es un mecanismo para implantar aplicaciones en el lado del servidor. Reciben peticiones y mandan resultados en HTTP, siendo el formato más común de salida una página HTML o un archivo XML, pero en el caso de este proyecto se utilizará JSON ya que es menos pesado que un archivo XML, además Google recomienda usar como buena práctica este tipo de formato en Android. El apartado 2.7 se hará una explicación más detallada de JSON.

Las tareas asignadas a un *Servlet* son las siguientes:

- **Leer los datos enviados por el cliente:** el usuario final enviará los datos al *Servlet* mediante un formulario HTML en una página Web o una solicitud HTTP, en la que, además, se incluye información como: *cookies*, tipo de datos que se envían, tamaño de estos, etc.
- **Generar la respuesta:** este proceso requiere que el *Servlet* se comunique, en el caso que nos concierne, con la base de datos y obtener los datos deseados. También podría ejecutar un RMI1, invocar un servicio Web, o procesar la respuesta directamente.
- **Enviar la respuesta:** los datos pueden ser enviados en una gran variedad de formatos entre los que podemos citar texto, binario o incluso un formato comprimido.

Los *Servlets* son una mejora al modelo anteriormente utilizado en la Web, CGI (Common Gateway Interface). Las ventajas respecto a CGI son las siguientes:

- **Multiplataforma**, ya que es Java.
- **Menor consumo de recursos**, cada petición genera un hilo dentro de un proceso. Frente a la arquitectura CGI, donde cada petición genera un proceso.
- **Más seguros**, pueden programarse de tal forma que se ejecuten en una zona de memoria con restricciones de seguridad para no acceder a determinados espacios de archivos del servidor o determinadas aplicaciones.

Los *Servlets* HTTP derivan de la clase `javax.servlet.http.HttpServlet`. La clase HttpServlet sobrescribe el método service() de forma que puede manejar diferentes tipos de solicitudes HTTP: DELETE, GET, OPTIONS, POST, PUT, y TRACE. Por cada uno de estos tipos de solicitud, la clase HttpServlet proporciona su correspondiente método doXXX().

Aunque se puede sobrescribir (redefinir) el método service() en la clase *Servlet*, raramente hay alguna necesidad de hacerlo. Más frecuentemente se requerirá redefinir métodos *doXXX*() individuales. Para la mayoría de las aplicaciones, será necesario sobrescribir únicamente los métodos *doPost()* y *doGet(),* ya que ellos manejan normalmente los datos enviados por un formulario o una aplicación.

### 2.3.1 Arquitectura de un Servlet

Todos los *Servlets* implementan una interfaz directamente, que es el principal componente que proporciona la API, llamada **HttpServlet**. Esta interfaz está provista de métodos que manipulan a los *Servlets* y la comunicación con sus clientes.

Cuando un *Servlet* es llamado desde un cliente, este recibe dos objetos:

- **HttpServletRequest**: Esta interfaz se encarga de la comunicación desde el cliente al servidor. Permite al *Servlet* acceder a información como, los nombres de parámetros pasados por el cliente, el protocolo usado por el cliente, y los nombres de los host remoto que hacen la solicitud y el servidor que la recibe.
- **HttpServletResponse**: Esta interfaz atiende la comunicación desde *Servlet* al cliente, es decir, proporciona al *Servlet* los métodos para contestarle al cliente. Permite configurar la forma de salida de los datos.

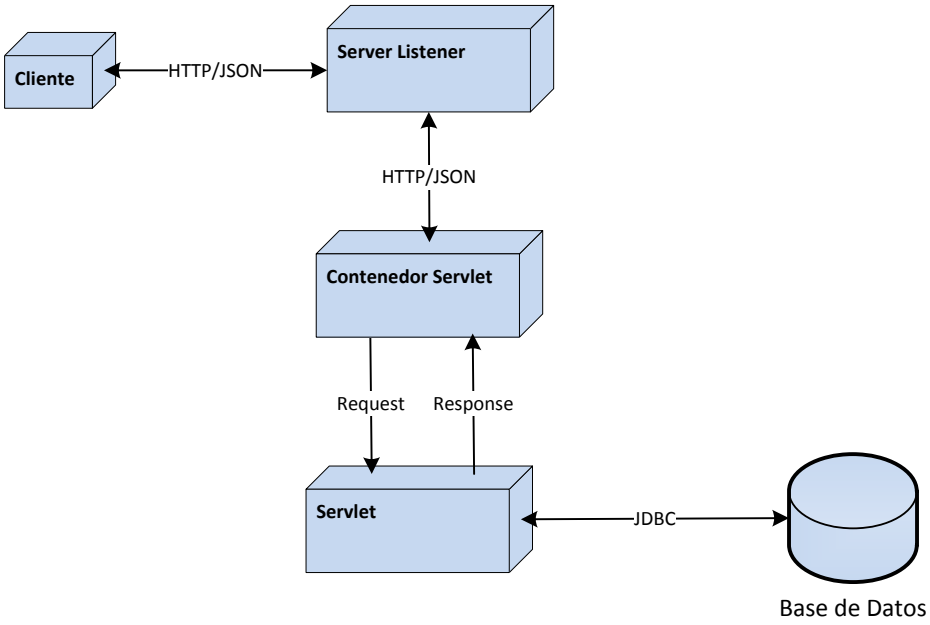


Figura 2.4: Arquitectura de un Servlet

## 2.4 Bases de datos

Una base de datos (BBDD) [5] es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. Dentro de esta definición global, nos centraremos en las bases de datos en formato digital que permiten almacenar una gran variedad de tipos de datos.

Para poder manipular y gestionar las BBDD se necesitan de unos programas denominados sistemas gestores de bases de datos (SGBD) que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

De entre los diferentes tipos de BBDD la más usual, y la que se utiliza en el PFC, es la BBDD del tipo relacional. Las BBDD relacionales de basan en el uso de relaciones o tablas que a su vez, están compuestas por registros (las filas de la tabla). La principal ventaja de estas bases de datos es que no importa el orden en el que se almacenan los datos. Para poder introducir datos o extraerlos se utilizan consultas a la BBDD. También permite ordenarlas tomando algún campo de la tabla como criterio.

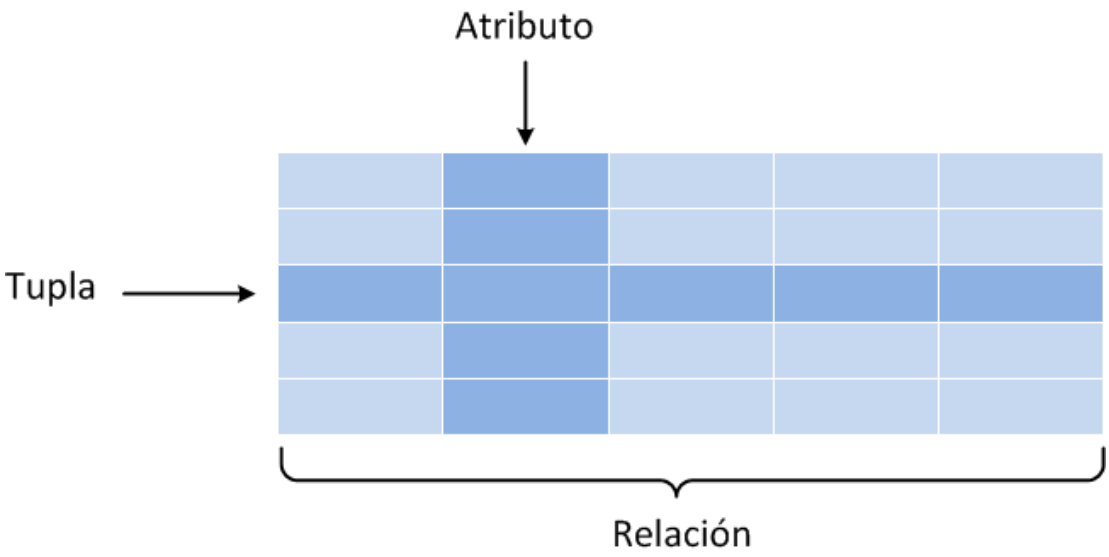


Figura 2.5: Estructura de una base de datos relacional

**Ventajas** de las BBDD de tipo relacional:

- Garantizan herramientas para **evitar la duplicidad** de registros, a través de campo clave o llaves.
- Garantiza la **integridad referencial**, es decir, al eliminar un registro elimina todos los registros relacionados dependientes.
- Favorece la **normalización** por ser más comprensible y aplicable.

Entre los **puntos fuertes** de una BBDD están la facilitación de la manipulación de **grandes volúmenes** de datos, estas facilidades bajan notablemente los tiempos de desarrollo y aumentan la calidad del sistema.

## 2.5 SQL

El **lenguaje de consulta estructurado** SQL (*Structured Query Language*) [5] es un lenguaje declarativo de acceso a BBDD relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar, de una forma sencilla, información de interés de una base de datos, así como también hacer cambios sobre la misma.

La primera versión de SQL fue desarrollada en IBM por *Donald Chamberlin* y *Raymond Boyce* a principios de los 70. Esta versión, inicial denominada *SEQUEL*, fue diseñada para manipular una BBDD relacional original de IBM (*System R*). IBM patentó esta versión de SQL en 1985, mientras que el lenguaje SQL no fue estandarizado hasta 1986, por el *ANSI*.

El lenguaje SQL se divide en varios subelementos, y son los siguientes:

- **Statements**: Tienen un efecto persistente en esquema o en los datos.
- **Queries**: Retornan datos basándose en un criterio.
- **Expressions**: Producen valores para las tablas.
- **Predicates**: Especifican condiciones que pueden ser evaluadas por SQL para limitar los efectos de los statements o queries.
- **Clauses**: Constituyen componentes de los statements o queries.

La operación más común en SQL es la query o petición, la cual se realiza mediante la palabra clave SELECT:

```
SELECT * FROM Users WHERE name='Eduardo'
```

Entre sus **puntos fuertes** se encuentra que al ser un lenguaje declarativo de "alto nivel" gracias a su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación y la orientación a objetos, pudiendo realizar en una sola sentencia varias operaciones de lenguaje a bajo nivel.

Entre sus **puntos débiles** se tiene que al ser un lenguaje declarativo especifica qué es lo que se quiere y no cómo conseguirlo, por lo que una sentencia no establece explícitamente un orden de ejecución.

Además hay otros aspectos que afectan negativamente a SQL:

- Las **diferentes implementaciones** de SQL son normalmente incompatibles entre desarrolladores.
- La sintaxis de SQL puede ser **compleja** dando a lugar a peticiones erróneas y que alteren de manera negativa la BBDD.

## 2.6 JDBC

Al programar aplicaciones para solucionar problemas del mundo real, habitualmente aparecen requisitos en los que se necesita almacenar, navegar y modificar datos. En un entorno empresarial, es necesario trabajar con los DBMS (Database Management Systems) para manejar las grandes cantidades de datos que se tiene. Sin embargo, la interacción con los DBMS (en adelante, simplemente se referirá los DBMS como bases de datos) no es realmente trivial. Exisiten numeroros sistemas de bases de datos empresariales y de código abierto y son diferentes unos de otros otros: DB2, SQL Server, MySQL, Oracle, y muchos más. Esta heterogeneidad de bases de datos populares hace que sea difícil escribir el código que se puede utilizar con cualquier base de datos. Para resolver estos problemas y hacer la vida fácil, Java ofrece JDBC (Java Database Connectivity). JDBC es un conjunto de API's proporcionadas por Java para interactuar mediante programación con diversas bases de datos.

A un alto nivel, interactuar con una base de datos incluye los siguientes pasos:

- El establecimiento de una conexión a una base de datos.
- Ejecución de consultas SQL para recuperar, crear o modificar una base de datos.
- Cierre de la conexión a la base de datos.

Java proporciona un conjunto de API's (es decir, JDBC [6]) para llevar a cabo estas actividades con bases de datos. En otras palabras, puede utilizarse JDBC para establecer una conexión con una base de datos, ejecutar una consulta SQL y cerrar la conexión con la base de datos. La belleza de JDBC es que no se está escribiendo un programa para una base de datos específica. JDBC crea un acoplamiento débil entre el programa Java y el tipo de base de datos utilizada. Por ejemplo, algunas bases de datos pueden diferir en la forma en que establecen una conexión (por ejemplo, el nombre de la API puede diferir según la base de datos). JDBC oculta toda la heterogeneidad de estas bases de datos y ofrece un único conjunto de API's para interactuar con todos los tipos de bases de datos.

### 2.6.1 Arquitectura de JDBC

Se van a examinar los componentes vitales de JDBC y cómo estos componentes trabajan juntos para lograr una perfecta integración con algunas bases de datos. Una arquitectura simplificada de JDBC se representa gráficamente en la Figura 2.6. Una aplicación Java utiliza las API de JDBC para interactuar según la base de datos. La API de JDBC interactúa con el gestor de controladores JDBC para conectar de forma transparente y realizar diversas actividades de base de datos con diferentes tipos de bases de datos. El gestor de controladores JDBC utiliza varios controladores JDBC para conectarse a sus DBMS específicos.

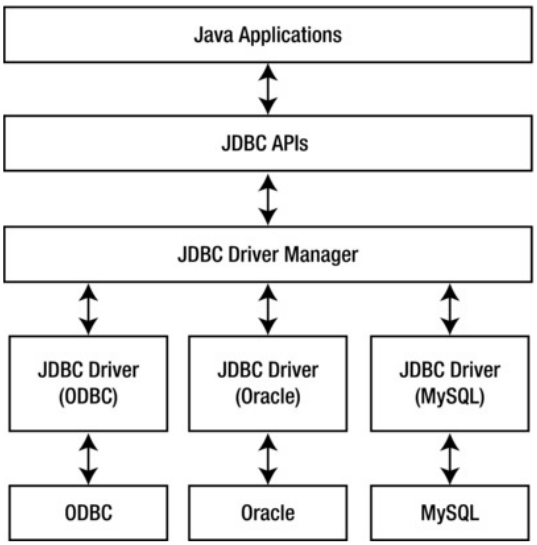


Figura 2.6: Arquitectura JDBC

En este contexto, los controladores JDBC (JDBC Drivers) y el gestor de controladores juegan un papel clave en la consecución del objetivo de JDBC (Java Database Connectivity). Los controladores JDBC están específicamente diseñados para interactuar con sus respectivas DBMS. El gestor de controladores funciona como un directorio de drivers JDBC, es decir, que mantiene una lista de fuentes de datos disponibles y sus controladores. El gestor de controladores elige un controlador adecuado para comunicar a los respectivos DBMS. Puede manejar múltiples drivers simultáneos conectados a sus respectivas fuentes de datos.

Se puede observar que la complejidad de las interacciones heterogéneas se delega en el gestor de controladores JDBC y los drivers JDBC, por lo que todos los detalles y las complicaciones están ocultas por la API JDBC para el desarrollador de la aplicación.

### 2.6.2 Arquitectura en dos y tres niveles

En términos generales, las arquitecturas de JDBC pueden verse en dos configuraciones principales: dos niveles y tres niveles. En una configuración de dos niveles, una aplicación Java junto con un controlador JDBC constituye el primer nivel como un cliente. Por otro lado, la base de datos funciona como un servidor que cumple los requisitos enviados desde los clientes. Por lo general, la base de datos reside en una máquina diferente conectada a través la red.

El primer nivel en una configuración de tres niveles es una aplicación ligera de Java (un Applet, por ejemplo) que se comunica con el servidor de aplicaciones (el segundo nivel). El servidor de aplicaciones, a su vez reenvía la petición a la base de datos (el tercer nivel). En esta configuración, el nivel medio juega un papel vital ya que el comportamiento de la configuración puede ser ajustado en base a los requisitos (por ejemplo, la aplicación de control de acceso).

### 2.6.3 Tipos de drivers JDBC

Ya hay bastantes tipos de controladores JDBC que han surgido, y que están siendo utilizados por la industria en función de las preferencias y necesidades. Se pueden clasificar sobre la base de la tecnología utilizada para comunicar con la respectiva DBMS. El tipo de estos drivers puede desempeñar un papel crítico en la selección de un DBMS apropiado para una aplicación Java. Hay cuatro tipos de controladores JDBC:

- **JDBC-ODBC bridge drivers (tipo 1):** ODBC (Open Database Connectivity), es una API de middleware portable escrita en C para acceder a bases de datos. La primera categoría corresponde a los conductores que están diseñados para trabajar con los controladores ODBC, juegan el papel de un puente desde una aplicación Java para un controlador ODBC.
- **Driver de API nativo (tipo 2):** Los controladores de bases de datos que pertenecen a esta categoría usan las bibliotecas de cliente de una base de datos específica y convierten las llamadas JDBC a llamadas de bases de datos nativas.

- **Driver de protocolo de red (tipo 3):** Este tipo de controlador de bases de datos implementa una arquitectura de tres niveles donde las llamadas JDBC se convierten en llamadas de bases de datos nativas a través de una aplicación middleware. En otras palabras, el controlador llama a la base de datos middleware el middleware en realidad convierte llamadas JDBC a las llamadas nativas específicas de bases de datos. Típicamente, el controlador se implementa en Java, que no requiere ninguna otra aplicación cliente en el lado del cliente, por lo que podrían ser empleados en aplicaciones basadas en Internet. Sin embargo, estos drivers son típicamente más lento que bridge drivers de tipo 2.
- **Driver de protocolo nativo (tipo 4):** Estos drivers se implementan en Java por completo, por lo que son independientes de la plataforma. Los controladores que pertenecen a esta categoría hacen directamente llamadas específicas de bases de datos en la red sin ningún tipo de apoyo a las bibliotecas del lado del cliente adicionales.

## 2.7 ORM

El mapeo objeto-relacional (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas ORM) [7] es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan el mapeo relacional de objetos, aunque algunos programadores prefieren crear sus propias herramientas ORM.

En la programación orientada a objetos, de las tareas de gestión de datos son implementadas generalmente por la manipulación de objetos, los cuales son casi siempre valores no escalares. Para ilustrarlo, considere el ejemplo de una entrada en una libreta de direcciones, que representa a una sola persona con cero o más números telefónicos y cero o más direcciones. En una implementación orientada a objetos, esto puede ser modelado por un "objeto persona" con "campos" que almacenan los datos de dicha entrada: el nombre de la persona, una lista de números telefónicos y una lista de direcciones. La lista de números telefónicos estaría compuesta por "objetos de números telefónicos" y así sucesivamente. La entrada de la libreta de direcciones es tratada como un valor único por el lenguaje de programación (puede ser referenciada por una sola variable, por ejemplo). Se pueden asociar varios métodos al objeto, como uno que devuelva el número telefónico preferido, la dirección de su casa, etc.

Sin embargo, muchos productos populares de base de datos, como los Sistemas de Gestión de Bases de Datos SQL, solamente pueden almacenar y manipular valores escalares como enteros y cadenas, organizados en tablas normalizadas. El programador debe convertir los valores de los objetos en grupos de valores simples para almacenarlos en la base de datos (y volverlos a convertir luego de recuperarlos de la base de datos), o usar sólo valores

escalares simples en el programa. El mapeo objeto-relacional es utilizado para implementar la primera aproximación.

El núcleo del problema reside en traducir estos objetos a formas que puedan ser almacenadas en la base de datos para recuperarlas fácilmente, mientras se preservan las propiedades de los objetos y sus relaciones; estos objetos se dice entonces que son persistentes.

En el caso particular de Android, se utiliza una técnica ORM para leer la base de datos SQLite, a través de la cual se generan unos objetos Java denominados cursores, que contienen la información pedida a la BBDD.

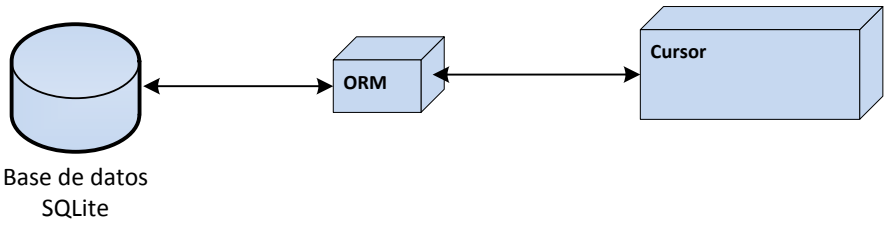


Figura 2.7: ORM en Android

## 2.8 HTTP

El protocolo de transferencia de Hipertexto HTTP (*HyperText Transfer Protocol*) [8] es el protocolo que define el formato y la secuencia de los mensajes transmitidos entre el navegador (cliente) y el servidor Web. HTTP fue desarrollado por el consorcio W3C y la IETF, colaboración que culminó en 1999 con la publicación de una serie de *RFCs*, siendo el más importante de ellos el *RFC2616*, que define la versión 1.1 de HTTP.

HTTP define la sintaxis y la semántica que utilizan los terminales de la arquitectura web (clientes, servidores y proxies) para comunicarse.

Es un protocolo orientado a transacciones y sigue el esquema request-response (petición-respuesta) entre un terminal cliente y un terminal servidor. Al cliente que efectúa la petición se le conoce como “user agent”, a la información transmitida se le llama recurso y se la identifica mediante un URL. Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, etc.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. Esto simplifica el diseño de los servidores y ha permitido a los ingenieros desarrollar servidores web de alto rendimiento que pueden manejar cientos de conexiones TCP simultáneas. Sin embargo, el desarrollo de aplicaciones necesita

frecuentemente mantener su estado. Para esto se usan las cookies (HTTP) o las sesiones en los *Servlets* (programas Java en el servidor), que se introducirán más adelante.

Las *cookies* almacenan la información en el terminal cliente, esto les permite a las aplicaciones web interpretar la noción de sesión, y también permite rastrear usuarios ya que las cookies se pueden almacenar en el terminal cliente por un periodo indeterminado.

Los *Servlets* son programas escritos en Java y que se ejecutan en el servidor, permite guardar más cantidad de información, a través de un método específico para ello, pero por periodos más pequeños de tiempo, es decir, después de cerrar la conexión almacenará la información el tiempo que este programado por el desarrollador .

El formato de las **peticiones** HTTP se puede ver en la figura 2.8, están formadas por 3 campos principales:

- **Línea de petición**: indica el método utilizado, la URL y la versión de HTTP
- **Líneas de cabecera:** Indica las cabeceras.
- **Cuerpo de entidad**: Es el cuerpo del mensaje. Entre el cuerpo del mensaje y las dos anteriores se introduce una línea en blanco.

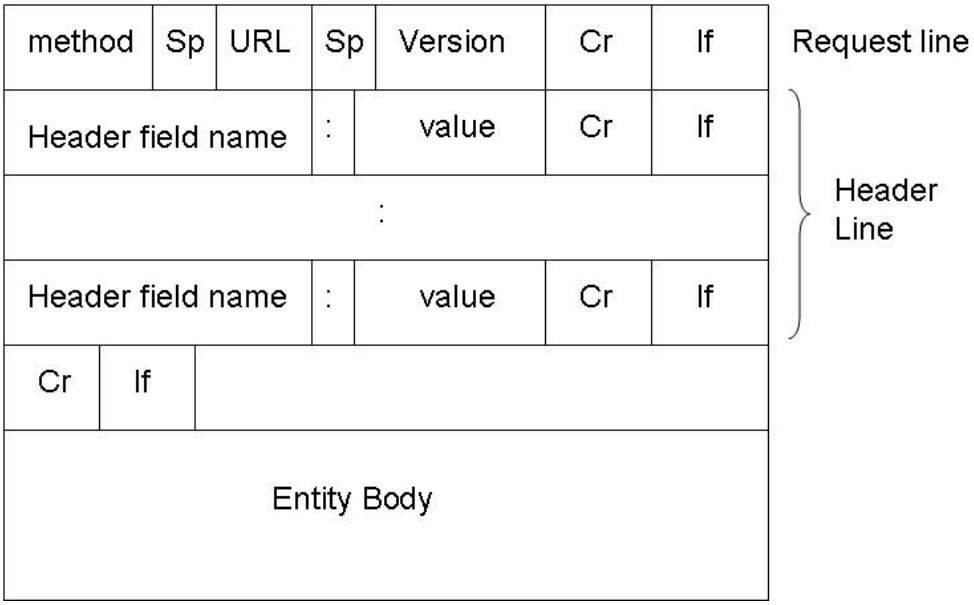


Figura 2.8: Formato general de una petición HTTP

HTTP define 8 métodos para realizar determinadas acciones **pero los más utilizados** son 3:

- **HEAD**: Solicita una petición HTTP pero que **no incluye el objeto pedido**. Es utilizado por desarrolladores para depurar aplicaciones.
- **GET:** Solicita una representación del recurso solicitado con el cuerpo del mensaje incluido. Si envía algún tipo de información esta se envía en la URL en **forma de parámetro**.

- **POST:** Envía información para que sea procesada en el recurso identificado. La información **se incluye** en el cuerpo de la petición.

HTTP tiene entre los **puntos fuertes** que es un protocolo sin estado, lo que permite que los host clientes no necesiten retener información entre peticiones. HTTP permite conexiones seguras y persistentes mediante las cuales se reduce el tiempo de espera ya que el cliente no debe renegociar las sucesivas conexiones después de la primera.

Entre los **puntos débiles** tiene que al ser un protocolo sin estado los desarrolladores deben pensar formas de guardar el estado. Una de las más comunes son las *cookies*. Dicho método también deja al descubierto determinados agujeros de seguridad.

## 2.9 JSON

JSON (JavaScript Object Notation) [9] es una forma muy ligera de intercambiar información y fácilmente legible y generable tanto para una máquina como para los humanos. Es un formato de texto completamente independiente del lenguaje, pero usa una sintaxis familiar para programadores. Estas propiedades lo convierten en un lenguaje ideal para intercambiar datos.

JSON utiliza dos estructuras: objetos (claves nombre y valor) y arrays (listas ordenadas de valores). Un valor puede ser un String, número, otro objeto, un array, un booleano o nulo. Su representación es la siguiente:

- **Objeto:**

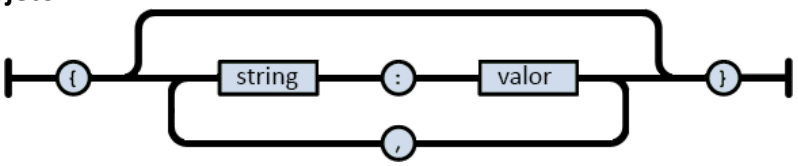


Figura 2.9: Objeto JSON

- **Array:**

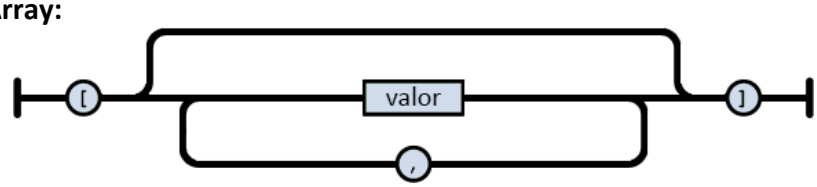


Figura 2.10: Array JSON

- **Valor:**

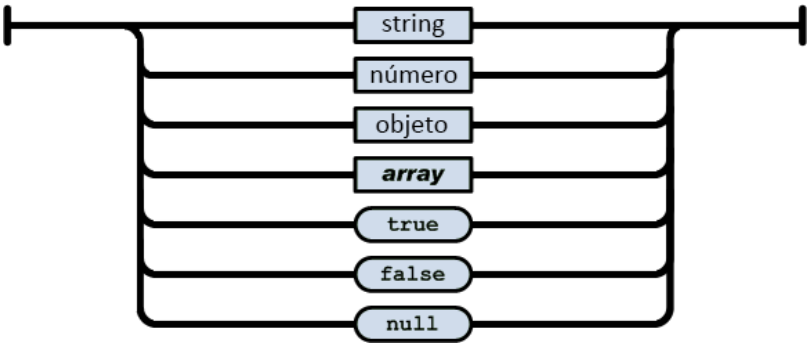


Figura 2.11: Posibles valores JSON

En el siguiente ejemplo, se puede ver cómo el servidor enviaría los datos a un usuario:

```
[{"Ubicar":"true","Estado":"Disponible","Latitud":"20.451078","Numero":"60
0000004","Precision":"0","Altitud":"0.0","Longitud":"6.71244"},{"Ubicar":"
true","Estado":"Disponible","Latitud":"20.4510306","Numero":"600000001","P
recision":"36.164","Altitud":"0.0","Longitud":"31.7125155"},{"Ubicar":"tru
e","Estado":"Probando","Latitud":"10.4509188","Numero":"600000009","Precis
ion":"10","Altitud":"70","Longitud":"39.7129586"}]
```

## 2.10 Android

Android es una pila de software de código abierto para dispositivos móviles que incluye sistema operativo, *middleware* y aplicaciones básicas. Google Inc. compró la empresa desarrolladora inicial del software, Android Inc., en el 2005.

El sistema operativo de Android está basado en una versión modificada del kernel de Linux. Google y otros miembros de la *Open Handset Alliance* colaboran en el desarrollo y lanzamiento de Android.

En el cuarto trimestre del 2010, el sistema operativo Android fue la plataforma de smartphones más vendida del mundo, destronando al Symbian de Nokia de la primera posición por primera vez en 10 años. Otras fuentes indican que Symbian estaba aun ligeramente por delante en ventas, si se tenían en cuenta algunos teléfonos de modelos antiguos Symbian de Nokia.

Android tiene una gran comunidad de desarrolladores programando aplicaciones que extienden la funcionalidad de los dispositivos, programan principalmente en el lenguaje Java y controlan el dispositivo mediante librerías Java desarrolladas por Google.

La llegada de la distribución Android, en noviembre del 2007, se anunció con la fundación de la *Open Handset Alliance*. Google lanzó la mayor parte del código Android bajo la licencia Apache, una licencia de software libre y código abierto.

La pila de software del sistema operativo Android consiste en aplicaciones Java que se ejecutan en un *framework* de aplicaciones basado en Java y orientado a objetos encima de librerías base Java, que se ejecutan en una máquina virtual *Dalvik*.

También hay librerías escritas en C que incluyen el gestor de superficie, el *OpenCore Media Framework*, el sistema gestor de base de datos relacional *SQLite*, la API gráfica *3D OpenGL ES 2.0*, el *WebKit layout engine*, el motor gráfico *SGL*, *SSL*, y *Bionic libc*.

El sistema operativo Android consiste en doce millones de líneas de código que incluyen tres millones de líneas de XML, 2,8 millones de líneas de C, 2,1 millones de líneas de Java y 1,75 millones de líneas de C++.

### 2.10.1 Breve historia

En octubre del 2003, Andy Rubin, Rich Miner y otros fundaron Android Inc. en Palo Alto, California (EE. UU). En palabras de Rubin, el objetivo era desarrollar "dispositivos móviles más elegantes que tuvieran más en cuenta la localización y las preferencias de sus dueños".

Entre otros empleados iniciales importantes se incluyen Andy McFadden, que trabajó con Rubin en WebTV, y Chris White, que lideró el diseño y la interfaz de WebTV antes de ayudar a fundar Android.

Rubin, cofundador de Danger Inc., Miner, cofundador de Wildfire Communications Inc. y vicepresidente de tecnología e innovación en Orange, y los otros empleados iniciales llevaron una considerable experiencia en la industria inalámbrica a la compañía. A pesar de los logros obvios del pasado de los fundadores y de los primeros empleados, Android Inc. funcionó de forma reservada y simplemente admitió que estaba trabajando en software para teléfonos móviles.

En agosto del 2005, Google adquirió Android Inc. Los empleados principales de Android Inc., entre los que se encontraban Andy Rubin, Rich Miner y Chris White, permanecieron en la compañía después de la adquisición.

En Google, el equipo liderado por Rubin desarrolló una plataforma para dispositivos móviles basada en el kernel de Linux. Google puso en el mercado la plataforma para los fabricantes de dispositivos móviles y los operadores con la premisa de proporcionar un sistema flexible y actualizable. Google alineó a una serie de socios dedicados a componentes hardware y software e indicó a los operadores que estaba abierto a varios grados de cooperación por su parte.

En noviembre del 2007, se anunció a sí misma *Open Handset Alliance*, un consorcio de varias compañías, entre las que se encuentran *Texas Instruments, Broadcom Corporation,*

*Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel y T-Mobile*.

Excepto durante breves periodos de actualización, Android ha estado disponible bajo una licencia de software libre de código abierto desde octubre de 2008. Google publicó todo el código fuente (incluyendo las pilas de red y telefonía) bajo una licencia Apache. Google también mantiene pública la lista de problemas revisados para que cualquiera pueda verla y comentarla.

### 2.10.2 Arquitectura de Android

Como se muestra en el siguiente diagrama, los componentes que forman Android se agrupan en capas. Cada una de estas capas utiliza elementos de la capa inferior para realizar sus funciones. Por ese motivo, a este tipo de arquitectura se le denomina pila. Esta es la pila software de Android:

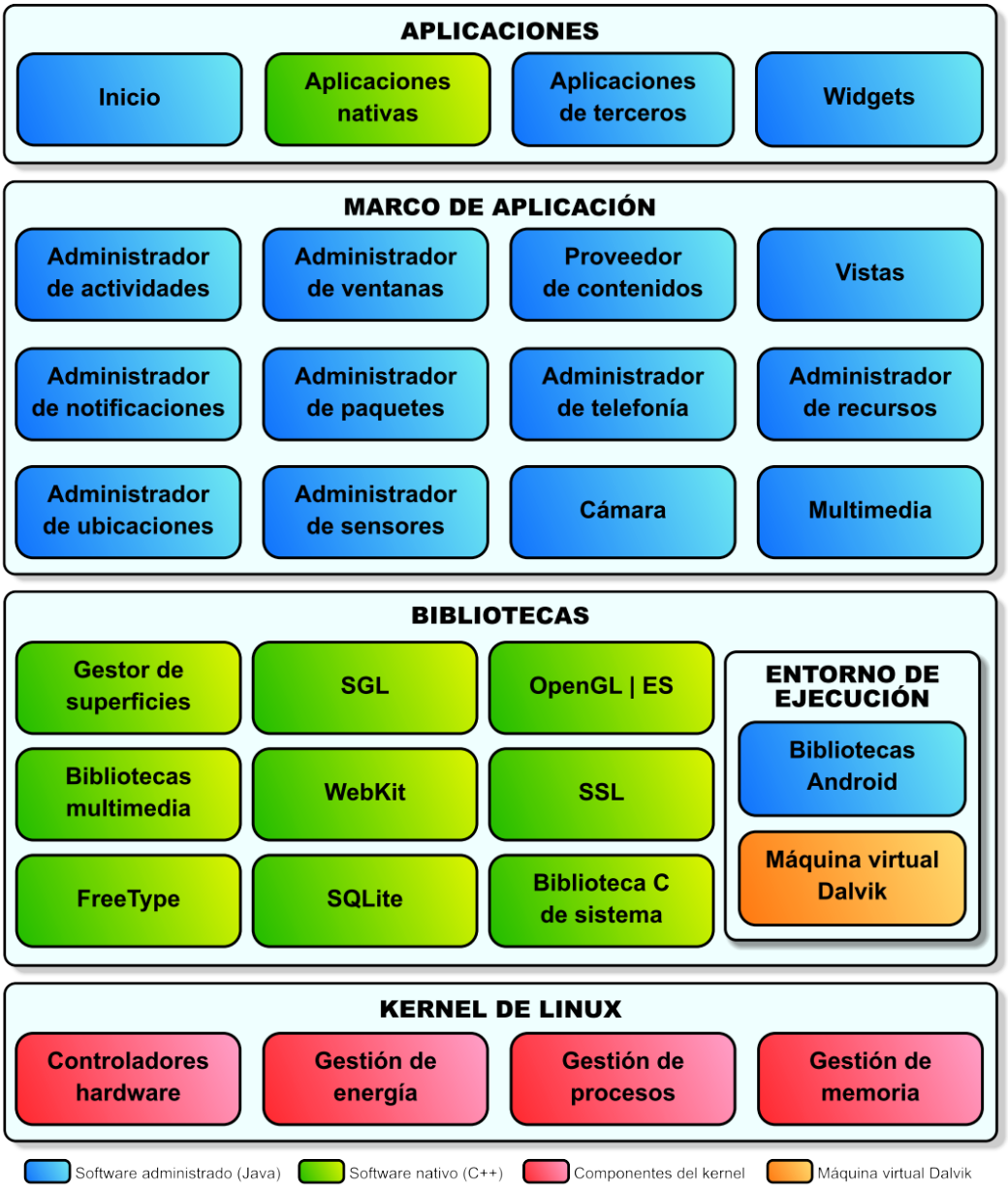


Figura 2.12: Arquitectura de Android

A continuación, se detallarán los diferentes componentes principales de la arquitectura de Android:

- **Núcleo Linux**

El núcleo de Android está basado en el *kernel* de Linux, en el que se han realizado algunas modificaciones para adaptarlo a las necesidades de Android. Este sirve como capa de abstracción del *hardware* al que tienen que acceder las aplicaciones. De esta manera, si se necesita un elemento *hardware*, la aplicación simplemente pedirá el recurso en genérico, sin tener que conocer el modelo exacto de *hardware*. Por ejemplo, si se quiere acceder a la

cámara, no importará el tipo, la definición ni el fabricante, simplemente se accederá a ella mediante el driver correspondiente.

- **Bibliotecas**

La capa que se sitúa justo sobre el *kernel* la componen las bibliotecas nativas de Android. Estas bibliotecas están escritas en C o C++ y compiladas para la arquitectura *hardware* específica del teléfono, tarea que realiza normalmente el fabricante, que también se encarga de instalarlas en el terminal antes de ponerlo a la venta. Su cometido es proporcionar funcionalidad a las aplicaciones (para tareas que se repiten con frecuencia), evitar tener que codificarlas cada vez y garantizar que se llevan a cabo de la forma más eficiente. Algunas bibliotecas son:

- **OpenGL**, motor gráfico 3D basado en la API de Open GL 1.0, 1.1 y 2.0.
- **Bibliotecas multimedia**: basadas en *OpenCORE*, permiten visualizar, reproducir e incluso grabar numerosos formatos de imagen, vídeo y audio.
- **WebKit**: motor web utilizado por el navegador (tanto como aplicación independiente como embebido en otras aplicaciones). Es el mismo motor que utilizan Google Chrome y Safari (el navegador de Apple, tanto en Mac como en el iPhone).
- **Secure Sockets Layer (SSL)**: proporciona seguridad al acceder a Internet por medio de criptografía.
- **SQLite**: motor de bases de datos relacionales, disponible para todas las aplicaciones.
- **Biblioteca C de sistema (libc)**: está basada en la implementación de Berkeley Software Distribution (BSD), pero optimizada para sistemas Linux embebidos. Proporciona funcionalidad básica para la ejecución de las aplicaciones.

- **Entorno de ejecución (Runtime de Android)**

El entorno de ejecución de Android, aunque se apoya en las bibliotecas enumeradas anteriormente, no se considera una capa en sí mismo, dado que también está formado por bibliotecas. En concreto, por las bibliotecas esenciales de Android, que incluyen la mayor parte de la funcionalidad de las bibliotecas habituales de Java, así como otras específicas de Android. Se basa en funcionalidades del *kernel* (como el *threading* o la gestión de memoria a bajo nivel).

El componente principal del entorno de ejecución de Android es la máquina virtual *Dalvik*, componente que ejecuta todas y cada una de las aplicaciones no nativas de Android. Las aplicaciones se codifican normalmente en Java y son compiladas, pero no para generar un ejecutable binario compatible con la arquitectura hardware específica del dispositivo Android. En lugar de eso, se compilan en un formato específico para la máquina virtual Dalvik, que es la que las ejecuta. Esto permite compilar una única vez las aplicaciones y distribuirlas ya compiladas con la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera cada aplicación.

No se debe confundir *Dalvik* con la máquina virtual Java, pues son máquinas virtuales distintas. De hecho, Google creó esta máquina virtual, entre otras cosas, para evitar problemas de licencias. Java es únicamente el lenguaje de programación, y los programas generados no son compatibles en cuanto a bytecode Java.

Los archivos generados para esta máquina virtual, archivos *.dex*, son muchos más compactos (hasta un 50%) que los generados para la máquina virtual Java tradicional. Para conseguir esto, Dalvik se basa en registros, en lugar de una pila para almacenar los datos, lo que requiere menos instrucciones. Esto permite ejecuciones más rápidas en un entorno con menos recursos.

Las aplicaciones Android se ejecutan cada una en su propia instancia de la máquina virtual Dalvik, de manera que se evitan interferencias entre ellas, y tienen acceso a todas las bibliotecas mencionadas antes y, mediante estas bibliotecas, al hardware y al resto de recursos gestionados por el kernel.

### ▪ Marco de aplicaciones

La siguiente capa la forman todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones y que, obviamente, se apoyan en las bibliotecas y en el entorno de ejecución que ya hemos detallado.

La mayoría de los componentes de esta capa son bibliotecas Java que acceden a los recursos mediante la máquina virtual *Dalvik*. Entre las más importantes se encuentran las siguientes:

- **Administrador de actividades (*activity manager*):** se encarga de controlar el ciclo de vida de las actividades y la propia pila de actividades. Sobre estas se puede hacer una metáfora con las ventanas de las aplicaciones de escritorio.

- **Administrador de ventanas (*Windows manager*):** se encarga de organizar lo que se muestra en pantalla y de crear, para ello, superficies que pueden ser "rellenadas" por las actividades.

- **Proveedor de contenidos (*content provider*):** permite encapsular un conjunto de datos que va a ser compartido entre aplicaciones y crear, de esa manera, una capa de abstracción que permita acceder a dichos datos sin perder el control sobre cómo se accede a la información. Por ejemplo, uno de los proveedores de contenido existentes permite a las aplicaciones acceder a los contactos almacenados en el teléfono. Esta biblioteca permite crear también proveedores propios para permitir que otras aplicaciones accedan esta información.

- **Vistas (*views*):** si antes se equiparaban las actividades con las ventanas de un sistema operativo de PC, las vistas se podrían equiparar con los controles que se suelen incluir dentro de esas ventanas. Android proporciona numerosas vistas con las que construir las interfaces de usuario: botones, cuadros de texto, listas, etc. También

proporciona otras más sofisticadas, como un navegador web o un visor de *Google Maps*.

- **Administrador de notificaciones (*notification manager*):** proporciona servicios para notificar al usuario cuando algo requiera su atención. Normalmente, las notificaciones se realizan mostrando alerta en la barra de estado, pero esta biblioteca también permite emitir sonidos, activar el vibrador o hacer pardear los LED del teléfono (si los tiene).

- **Administrador de paquetes (*package manager*):** las aplicaciones Android se distribuyen en paquetes (archivos .apk) que contienen tanto los archivos .dex como todos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación. Esta biblioteca permite obtener información sobre los paquetes actualmente instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes.

- **Administrador de telefonía (*telephony manager*):** proporciona acceso a la pila hardware de telefonía del dispositivo Android, si la tiene. Permite realizar llamadas o enviar y recibir SMS y MMS, aunque no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso (por motivos de seguridad).

- **Administrador de recursos (*resource manager*):** proporciona acceso a todos los elementos propios de una aplicación que se incluyen directamente en el código: cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos e, incluso, estructuraciones de las vistas dentro de una actividad (*layouts*). Permite gestionar esos elementos fuera del código de la aplicación y proporcionar diferentes versiones en función del idioma del dispositivo o la resolución de pantalla que tenga, para poder contrarrestar la fragmentación actual de dispositivos.

- **Administrador de ubicaciones (*location manager*):** permite determinar la posición geográfica del dispositivo Android (usando el GPS o las redes disponibles) y trabajar con mapas.

- **Administrador de sensores (*sensor manager*):** permite gestionar todos los sensores hardware disponibles en el dispositivo Android: acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, sensor de presión, sensor de proximidad, sensor de temperatura, etc.

- **Cámara:** proporciona acceso a las cámaras del dispositivo Android, tanto para tomar fotografías como para grabar vídeo.

- **Multimedia:** conjunto de bibliotecas que permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo

### ▪ Aplicaciones

La capa superior de esta pila software la forman, como no podría ser de otra forma, las aplicaciones. En este saco se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, tanto las nativas (programadas en C++) como las

administradas (programadas en Java), tanto las que vienen de serie con el dispositivo como las instaladas por el usuario.

Aquí está también la aplicación principal del sistema: inicio (*home*), también llamada a veces lanzador (*launcher*), porque es la que permite ejecutar otras aplicaciones proporcionando la lista de aplicaciones instaladas y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o, incluso, pequeñas aplicaciones incrustadas o *widgets*, que son también aplicaciones de esta capa.

El aspecto principal a tener en cuenta de esta arquitectura es que todas las aplicaciones (las nativas de Android, las que proporciona Google, las que incluye de serie el fabricante del teléfono o las que instala después el usuario) utilizan el mismo marco de aplicación para acceder a los servicios que proporciona el sistema operativo. Esto implica **dos cosas**: que podemos crear aplicaciones que usen los mismos recursos que usan las aplicaciones nativas (nada está reservado o inaccesible) y que podemos reemplazar cualquiera de las aplicaciones del teléfono por otra de nuestra elección.

Este es el verdadero potencial de Android y lo que lo diferencia de su competencia: el **control total** por parte del usuario del software que se ejecuta en su teléfono.

### 2.10.3 Android SDK

El kit de desarrollo de software (S*oftware Development Kit* o *SDK*) incluye un conjunto de herramientas de desarrollo, tales como un *debugger*, librerías, un emulador, documentación, código de ejemplo y tutoriales. Está soportado en S.O. Windows, Linux y Mac.

El entorno de desarrollo (*Integrated Development Environment* o *IDE*) oficialmente soportado es Eclipse conjuntamente con el *plugin* ADT proporcionado por Google.

Desde noviembre del 2007 hasta la actualidad han ido surgiendo nuevas actualizaciones del SDK siendo la última la 4.2, la cual proporciona una unificación para *tablets*, TV's, smartphones, *netbooks*, etc. La versión utilizada en este documento es la versión 2.3.3.

### 2.10.4 Partes de una aplicación Android

Dentro de una aplicación de Android hay varios componentes principales: Activitites, Intents, Services, Content Providers, Broadcast Receivers y Application Context [10]. Todas las aplicaciones de Android están formadas por algunos de estos elementos o combinaciones de ellos.

#### *2.10.4.1 Activity*

Una actividad es por lo general una sola pantalla que el usuario ve en el dispositivo al mismo tiempo. Una aplicación normalmente tiene múltiples actividades, y el usuario se mueve entre ellas. Como tal, las actividades son la parte más visible de la aplicación.

Al igual que un sitio web se compone de varias páginas, una aplicación Android consiste en múltiples actividades. Al igual que un sitio web tiene una "página de inicio", una aplicación para Android tiene una actividad "principal", por lo general la que se muestra por primera vez al iniciar la aplicación. Y al igual que un sitio web tiene que ofrecer algún tipo de navegación entre las distintas páginas, una aplicación para Android debería hacer lo mismo.

En la web, se puede saltar de una página de un sitio web a una página en otra. Del mismo modo, en Android, se podría estar mirando una actividad de una aplicación, pero poco después se puede iniciar otra actividad en una aplicación completamente independiente. Por ejemplo, si se está en la aplicación Contactos y se decide enviar un mensaje de texto a un amigo, se estaría poniendo en marcha la actividad para componer un mensaje de texto en la aplicación Mensajería.

### Ciclo de vida de una Actividad

Lanzar una actividad puede ser bastante costoso. Puede implicar la creación de un nuevo proceso de Linux, la asignación de memoria para todos los objetos de interfaz de usuario, inflando todos los objetos en la configuración XML, y la creación de toda la pantalla. Ya que se está haciendo mucho trabajo para poner en marcha una actividad, sería una pérdida de recursos destruirla una vez que el usuario sale de esa pantalla. Para evitar este desperdicio, el ciclo de vida de las actividades se gestione a través del Activity Manager (Administrador de Actividades).

Activity Manager se encarga de crear, destruir, y gestionar las actividades. Por ejemplo, cuando el usuario inicia una aplicación por primera vez, el Administrador de Actividades creará su actividad y la pondrá en la pantalla. Más tarde, cuando el usuario cambie de pantalla, el Administrador de Actividades trasladará la actividad previa a un lugar de almacenamiento. De esta manera, si el usuario quiere volver a una actividad anterior, se puede iniciar más rápidamente. Actividades más antiguas que el usuario no ha utilizado durante un tiempo serán destruidas con el fin de liberar más espacio para la que actualmente se encuentra activa. Este mecanismo está diseñado para ayudar a mejorar la velocidad de la interfaz de usuario y así mejorar la experiencia general del usuario.

Programar para Android es conceptualmente diferente a la programación de otros entornos. En Android, se responde más a ciertos cambios en el estado de la aplicación en lugar de realizar uno mismo el cambio. Se trata de un entorno administrado, basado en contenedores similar a la programación de Applets o Servlets Java. Por lo tanto, cuando se trata de un ciclo de vida de una actividad, no se puede llegar a decir en qué estado se encuentra la actividad, pero hay un montón de oportunidad de decir lo que sucede durante las transiciones de estado a estado. La Figura 2.13 muestra los estados por los que una actividad puede pasar.

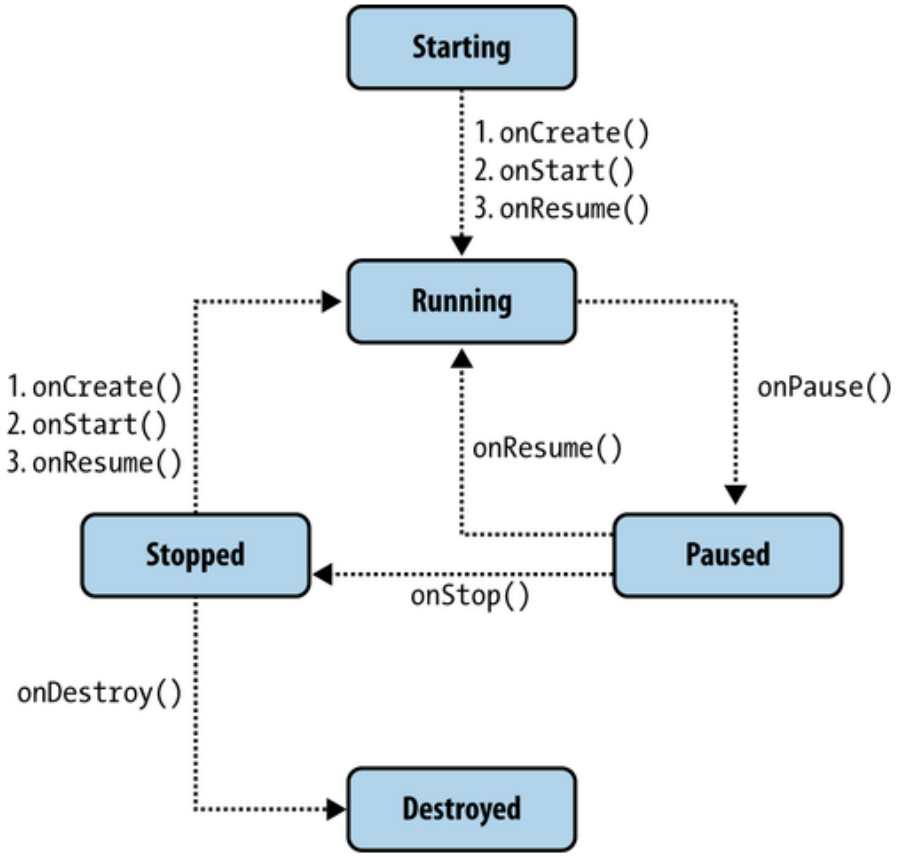


Figura 2.13: Ciclo de vida de una actividad

- **'Starting' – Estado de partida**

Cuando una actividad no existe en la memoria, que está en el estado de partida. Cuando está arrancando, la actividad pasará a través de un conjunto de métodos de devolución de llamada que el desarrollador tiene la oportunidad de llenar. Con el tiempo, la actividad estará en un estado de funcionamiento.

Hay que tener en cuenta que esta transición de un estado a otro de ejecución es una de las operaciones más costosas en términos de tiempo de cálculo, lo que también afecta directamente a la vida de la batería del dispositivo. Esta es la razón exacta por la que no se destruye automáticamente las actividades que ya no se muestran. El usuario podría querer volver a ellas, por lo que se mantendrán por un tiempo.

- **'Running' – Estado de ejecución**

La actividad en estado de ejecución es la que se encuentra actualmente en la pantalla e interactuando con el usuario. También se dice que esta actividad tiene el foco, lo que significa que todas las interacciones del usuario, tales como escribir, tocar la pantalla y hacer clic en los botones, son gestionados por esta actividad. Como tal, sólo hay una actividad en funcionamiento en un momento dado.

La actividad que se ejecuta es la que tiene prioridad en términos de conseguir la memoria y los recursos que necesita para funcionar lo más rápido posible. Esto se debe a que Android quiere asegurarse de que la actividad en ejecución es rápida y sensible para el usuario.

- **'Paused' – Estado pausado**

Cuando una actividad no tiene el foco (o sea, no interactúa con el usuario), pero sigue siendo visible en la pantalla, se dice que está en un estado de pausa. Esto no es un escenario típico, porque la pantalla del dispositivo es generalmente pequeña, y una actividad o bien está ocupando toda la pantalla o nada en absoluto. A menudo se ve este caso con cuadros de diálogo que se presentan delante de una actividad, haciendo que ésta se pause. Todas las actividades pasan por un estado de pausa cuando se van a detener.

Las actividades pausadas siguen teniendo alta prioridad en términos de conseguir la memoria y otros recursos. Esto es porque son visibles y no pueden ser eliminadas de la pantalla sin que se vea muy extraña para el usuario.

- **'Stopped' – Estado parado**

Cuando una actividad no es visible, pero todavía está en la memoria, se dice que está detenida. Una actividad detenida podrá traerse de nuevo al frente para convertirse en una actividad en ejecución otra vez. O bien, podría ser destruida y eliminarse de la memoria.

El sistema guarda actividades en un estado de detención, ya que es probable que el usuario todavía quiera volver a las actividades dentro de poco, y reiniciar una actividad detenida es mucho más barato que iniciar una actividad a partir de cero. Eso se debe a que ya se tienen todos los objetos cargados en la memoria y simplemente se tienen que llevar hasta el primer plano.

Las actividades detenidas se pueden eliminar de la memoria en cualquier momento.

- **'Destroyed' – Estado destruido**

Una actividad destruida ya no está en la memoria. El gestor de actividad decidió que ya no se necesitaba esta actividad y la ha eliminado. Antes de que se destruya la actividad, puede realizar ciertas acciones, como salvar cualquier información no guardada. Sin embargo, no hay garantía de que la actividad se detenga antes de ser destruida. Es posible que una actividad pausada sea destruida también. Por este motivo, es mejor hacer un trabajo importante, como salvar los datos no guardados, cuando se pause en lugar de cuando se vaya a destruir.

El hecho de que una actividad esté en estado de ejecución no quiere decir que esté haciendo mucho. Podría estar simplemente y esperando la entrada del usuario. Del mismo modo, una actividad en un estado de detención no está necesariamente haciendo nada. Los nombres de los estados principalmente se refieren a la forma de la actividad con respecto a la entrada del usuario, en otras palabras, si una actividad es visible, si tiene foco, o no es visible en absoluto.

### *2.10.4.2 Intents*

Los Intents son mensajes que se envían entre los principales bloques de la aplicación. Además, ponen en marcha una actividad, le indica a un servicio que se inicie o detenga, o son simplemente broadcasts. Los Intents son asíncronos, es decir, el código que los envía no tiene que esperar a que se completen.

Un Intent puede ser explícito o implícito. En un Intent explícito, el remitente explica claramente qué componente específico debe estar en el extremo receptor. En un Intent implícito, el remitente especifica el tipo de receptor. Por ejemplo, una actividad podría enviar un Intent diciendo que sólo quiere a alguien para abrir una página web. En ese caso, cualquier aplicación que sea capaz de abrir una página web puede "competir" para completar esta acción.

Cuando haya competencia entre aplicaciones, el sistema pedirá cuál desea utilizar para realizar una acción determinada. También se puede configurar una aplicación por defecto. Este mecanismo funciona de manera muy similar a un entorno de escritorio, por ejemplo, cuando se ha descargado Firefox o Chrome para reemplazar el navegador por defecto Internet Explorer o Safari.

Este tipo de mensajería permite al usuario sustituir cualquier aplicación en el sistema con una personalizada. Por ejemplo, es posible que desee descargar una aplicación SMS diferente u otro navegador para reemplazar los existentes. La figura 2.14 muestra cómo se pueden utilizar los Intents para "saltar" entre las diversas actividades, en la misma aplicación o en otra aplicación todos juntos.

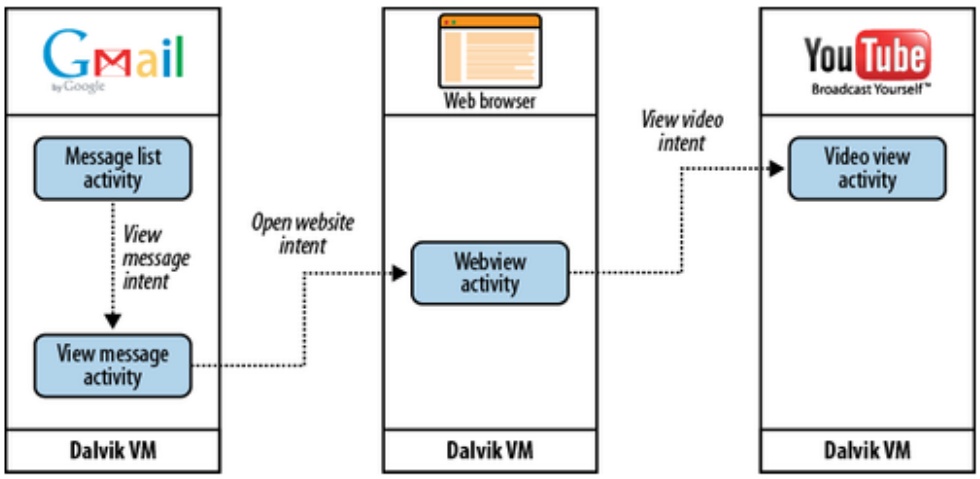


Figura 2.14: Intents

### *2.10.4.3 Service*

Los servicios se ejecutan en segundo plano y no tienen ningún componente de interfaz de usuario. Pueden realizar las mismas acciones que actividades, pero sin ninguna interfaz gráfica. Los servicios son útiles para las acciones que se quieran realizar durante un tiempo, con independencia de lo que esté en la pantalla. Por ejemplo, es posible que desee que el reproductor de música reproduzca música incluso mientras se esté navegando entre otras aplicaciones.

No hay que confundir los servicios de Android que son parte de una aplicación Android con servicios nativos de Linux, servidores, o demonios, que son un componente mucho más bajo nivel del sistema operativo.

Los servicios tienen un ciclo de vida mucho más simple que las actividades (ver la Figura 2.15). O se inicia un servicio o se detiene. Además, el ciclo de vida del servicio es más o menos controlada por el desarrollador, y no tanto por el sistema. Por lo tanto, los desarrolladores tienen que diseñar los servicios para que no consuman recursos compartidos innecesariamente, tales como la CPU y la batería.

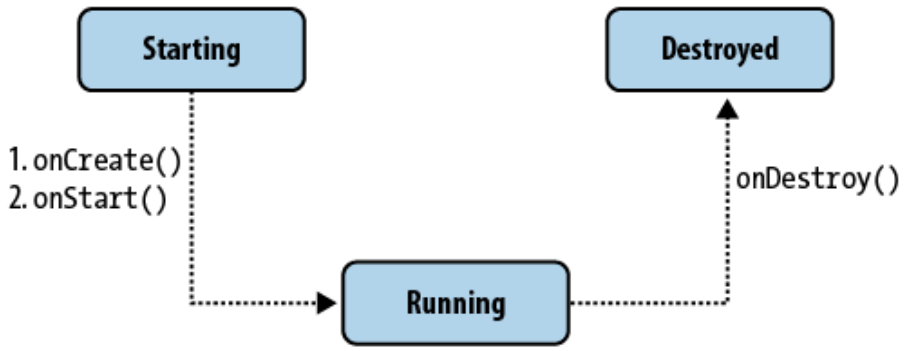


Figura 2.15: Ciclo de vida de un servicio

El hecho de que un servicio se ejecute en segundo plano, no significa que se ejecute en un hilo independiente. Si un servicio está realizando una tarea que toma tiempo para completarse (como realizar llamadas de red), normalmente se ejecutaría en un hilo independiente. De lo contrario, la interfaz de usuario se ejecutará mucho más lenta. En otras palabras, los servicios y las actividades se ejecutan en el mismo hilo de la aplicación principal, a menudo llamado el hilo de interfaz de usuario.

### *2.10.4.4 Content Providers*

Los Content providers (proveedores de contenido) son interfaces para el intercambio de datos entre aplicaciones. De forma predeterminada, Android ejecuta cada aplicación en su propia sandbox para que todos los datos que pertenecen a una aplicación estén totalmente aislados de otras aplicaciones en el sistema. Aunque se pueden pasar pequeñas cantidades de datos entre aplicaciones a través de los Intents, los content providers son mucho más adecuados para el intercambio de datos persistentes entre grandes conjuntos de datos. Como tal, la API de los content providers bien se adhiere al principio CRUD. La figura 2.16 ilustra cómo la interfaz CRUD del proveedor de contenidos perfora los límites de las aplicaciones y permite otras aplicaciones que se conecten a ella para compartir datos.

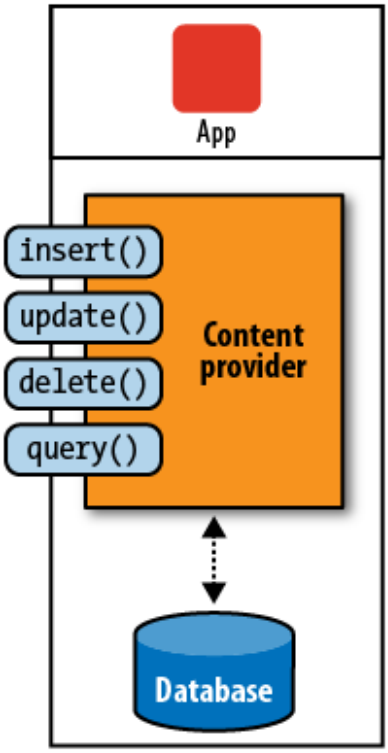


Figura 2.16: Content provider

El sistema de Android utiliza este mecanismo todo el tiempo. Por ejemplo, Contacts Provider es un proveedor de contenido que expone todos los datos de contacto de usuario a varias aplicaciones. Settings Provider expone la configuración del sistema para diversas aplicaciones, incluyendo la aplicación de ajustes incorporados. Media Store es responsable de almacenar y compartir diversos medios de comunicación, como fotos y música, a través de diversas aplicaciones. La Figura 2.17 ilustra cómo la aplicación Contacts utiliza el proveedor contactos, una aplicación totalmente independiente, para obtener datos sobre los contactos

de los usuarios. La propia aplicación Contacts no tiene ninguna información de contactos y Contacts Provider no tienen ninguna interfaz de usuario.

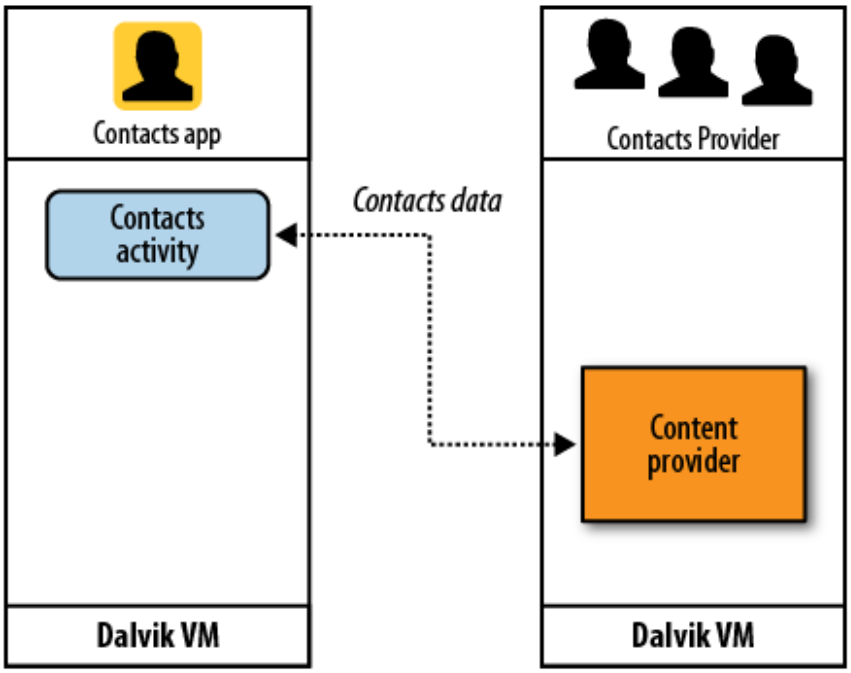


Figura 2.17: Aplicación Contacts utilizando Contacts Provider para obtener datos

Esta separación de almacenamiento de datos y la interfaz de usuario de la aplicación ofrece una mayor flexibilidad para juntar diversas partes del sistema. Por ejemplo, un usuario podría instalar una aplicación de agenda alternativa que utiliza los mismos datos que la aplicación Contacts predeterminada. O bien, se podrían instalar widgets en la pantalla de inicio que permitan cambiar fácilmente la configuración del sistema, tal como encender o apagar el Wi-Fi, Bluetooth o GPS. Muchos fabricantes de teléfonos aprovechan los content providers para añadir sus propias aplicaciones en la parte superior de Android estándar para mejorar la experiencia de usuario en general, tales como HTC Sense.

Los content providers son interfaces relativamente simples, con los métodos estándar insert(), update(), delete() y query() Estos métodos se parecen mucho a los métodos de base de datos estándar, por lo que es relativamente fácil de implementar un content provider como una aproximación a la base de datos. Una vez dicho esto, es mucho más probable que usen los content providers que programar uno propio.

### *2.10.4.5 Broadcast Receivers*

Los broadcast receivers (receptores de radiodifusión) son la implementación de Android de un amplio sistema de publicación / suscripción, o más precisamente, un patrón Observer. El

receiver es simplemente código de estado latente que se activa una vez que ocurre un evento al que está suscrito.

El sistema en sí transmite eventos todo el tiempo. Por ejemplo, cuando llega un SMS, se recibe una llamada, la batería está baja, el sistema se inicia, todos los eventos se transmiten, y cualquier número de receptores podrían ser activados por ellos.

En la aplicación de este proyecto, "Contacta!", se quiere iniciar el servicio de actualización una vez que el sistema se inicia. Para ello, se puede suscribir a la difusión que nos dice el sistema ha terminado de arrancar.

También pueden enviar broadcasts propios de una parte de la aplicación a otra, o de una aplicación totalmente diferente.

Los propios broadcast receivers no tienen ninguna representación visual, ni se ejecutan activamente en la memoria. Sin embargo, cuando se activan, llegan a ejecutar algún código, como el inicio de una actividad, un servicio, o alguna otra cosa.

#### *2.10.4.6 Application Context*

Hasta ahora se han visto las actividades, servicios, content providers y broadcast receivers. Juntos, constituyen una aplicación. Otra forma de decir esto es que viven dentro del mismo contexto de la aplicación.

El application context (contexto de aplicación) se refiere al entorno de la aplicación y el proceso en el que todos sus componentes se están ejecutando. Permite que las aplicaciones compartan los datos y los recursos entre los distintos bloques.

Un contexto de aplicación se crea cada vez que se inicia el primer componente de esta aplicación independientemente de que ese componente sea una actividad, servicio, o cualquier otra cosa. El contexto de aplicación está presente siempre y cuando la aplicación esté viva. Como tal, es independiente del ciclo de vida de las actividades. Se puede obtener una referencia al contexto llamando Context.getApplicationContext() o Activity.getApplication(). Hay que tener en cuenta que las actividades y servicios que ya son subclases de contexto, y como tales, heredan todos los métodos.

### 2.10.5 Tipos de aplicaciones

Un Android Package (.apk) es el fichero que contiene el código de la aplicación y sus recursos, y que posteriormente se instala en el dispositivo para poder ejecutar la aplicación.

#### *2.10.5.1 Tareas*

Una tarea en Android es lo que el usuario ve como una aplicación y el desarrollador ve como una o más Actividades donde el usuario interacciona y va pasando de vista en vista. Dentro de las tareas, una actividad toma el papel de punto de entrada (será la primera en mostrarse cuando se ejecute la aplicación) y las demás, si hay, formarán parte de la misma tarea, a la espera de ser instanciadas.

#### *2.10.5.2 Procesos*

En Android, los procesos se ejecutan a nivel de kernel y el usuario normalmente no tiene constancia de ellos. Todo el código de la aplicación se suele ejecutar en un proceso dedicado pero también se puede especificar si sólo se quiere que se ejecute en el proceso una determinada clase o componente de la aplicación. Los principales usos de los procesos son:

- Mejorar la estabilidad o seguridad de las aplicaciones.
- Reducir la sobrecarga de proceso ejecutando el código de múltiples aplicaciones en el mismo proceso.
- Ayudar al sistema a gestionar los recursos separando partes de código pesado en un proceso separado que puede ser eliminado independientemente de otras partes de la aplicación.

#### *2.10.5.3 Threads*

En lo referente a threads (hilos), Android evita la creación de hilos adicionales por parte de un proceso, manteniendo la aplicación en un sólo hilos al menos que no los cree la propia aplicación. Esto repercute de manera importante en las llamadas a instancias a Actividades y Servicios, ya que sólo pueden ser hechas por el hilo principal del proceso en el que están corriendo. Por otro lado, al no crearse un hilo por cada instancia, dichas instancias no deben realizar operaciones largas o bloqueantes cuando son llamadas, de lo contrario, bloquearían todos los demás componentes del proceso.

### 2.10.6 Recursos

Los recursos en Android son aquellos ficheros que pertenecen al proyecto. Pueden ser multimedia o estáticos. Según su organización en directorios y la información de contexto de la aplicación, se utilizarán para un tipo de dispositivo u otro. Dentro de los ficheros hay muchos ficheros XML que sirven para definir layouts, menús, cadenas de texto, estilos, etc.

Estos recursos constituyen una parte importante de la plataforma, ya que se usan en varias partes de la plataforma para objetivos distintos, pero tienen el mismo funcionamiento.

Por tanto, si sabemos cómo funcionan para un caso, sabremos cómo funcionarán para el resto, ya que el comportamiento será idéntico.

El principal objetivo de estos recursos es ahorrarle al desarrollador su gestión y delegar la elección de qué fichero utilizar a la plataforma Android. Esta es una manera de atacar la fragmentación en Android.

Para que estos recursos puedan ser interpretados por Android, deben estar organizados de una manera especial, siempre dentro del directorio res de la raíz del proyecto. A partir de aquí, tenemos varios subdirectorios, como por ejemplo:

- **Drawable:** son, básicamente, ficheros de imágenes o animaciones en XML.
- **Layout:** son ficheros que definen el diseño o distribución de los elementos en la pantalla.
- **Values:** son valores de cadenas de caracteres, enteros, colores, etc. que, por ejemplo, son candidatos a ser internacionalizados.
- Otros pueden ser **anim** (para ficheros XML de animaciones), **raw** (para ficheros arbitrario) o **menu** (para ficheros XML que definen un menú).

| Tipo | Descripción |
|---|---|
| **Anim** | Ficheros XML que son compilados en objetos de animaciones |
| **Drawable** | Ficheros con formato .png, .9.png, jpg que se compilan en un objeto bitmap |
| **Layout** | Ficheros XML que compilan en vistas de pantalla |
| **Values** | Ficheros XML que se compilan en varios tipos de recursos para definir, por ejemplo, arrays, strings, colores, etc. |
| **XML** | Ficheros XML arbitrarios que son compilados y pueden ser llamados en tiempo de ejecución |
| **Raw** | Ficheros arbitrarios que pueden ser copiados directamente al dispositivo. |

Tabla 2.1: Tipos de recursos en Android

Para que estos recursos puedan ser utilizados desde el código Java, además de los usos implícitos (como algunos layouts), la plataforma los transforma automáticamente a un fichero llamado R.java. Esta clase Java se regenera en cada cambio de los ficheros de recursos, de manera que todas las propiedades son identificadas de forma unívoca por atributos de la clase R de Java (por ejemplo, R.layout.main).

# 2.11 Google Maps

Mientras MapQuest fue el primer servicio de mapas en línea, ya no es el más popular. Este título ha sido tomado por un pariente recién llegado - Google Maps.

Como se puede sospechar por su nombre, Google Maps es un servicio de mapas en línea ofrecido por Google, la compañía del buscador. Google Maps ha sido pionera en la funcionalidad de mapas en línea, y continúa evolucionando sus mapas y otras funcionalidades.

Al igual que sus competidores, Google Maps ofrece una gran cantidad de servicios de mapas útiles, todo ello incluido en una interfaz fácil de usar. Permite generar mapas de cualquier dirección o lugar determinado, pero también se puede hacer clic y arrastrar los mapas para ver las secciones adyacentes, superponer al mapa información sobre imágenes de satélite de la zona determinada, ver las condiciones de tráfico locales, exhibir las empresas cercanas como una serie de chinchetas en el mapa, ver fotos a nivel de calle de un lugar determinado, e incluso generar indicaciones para ir de un lugar a cualquier otro.

A diferencia de sus competidores, Google Maps funciona como el propio buscador de Google, se introduce en la caja de búsqueda lo que se desea visualizar y éste lo muestra inmediatamente.

## 2.11.1 Breve historia

MapQuest había estado en la web durante casi una década antes de que Google entró en el negocio de los mapas. Lo que hoy conocemos como Google Maps tiene su inicio en 2003 como una aplicación de software de una empresa llamada Where2 Technologies. Google adquirió la compañía y su software de mapas en 2004, y se puso a trabajar su conversión desde una aplicación de software de escritorio tradicional a una basada en la web.

Una vez hecho, Google anunció su servicio de mapas en línea en febrero de 2005. Estuvo en beta pública durante seis meses antes de ser formalmente parte del servicio Google Local en octubre de ese año.

Así es, Google Maps se llamó originalmente Google Local-o, mejor dicho, era una característica complementaria al servicio Google Local. Google Local, que ya no existe (con ese nombre, de todos modos), era un servicio que ofrece listados de empresas locales-una especie de páginas amarillas de la web. Lo único que faltaba era agregar la funcionalidad de mapas a estos anuncios, que es lo que hizo Google.

Lo que Google descubrió, sin embargo, era que la gente estaba más interesada en la cartografía de lo que eran los listados locales. Para ello, Google cambió el énfasis y en abril de 2006, cambió el nombre entero tinglado de Google Maps. A día de hoy, puede utilizar Google Maps para buscar negocios en torno a una dirección o ubicación específica, que es la herencia de Google Local.

### 2.11.2 Tipos de mapas

La vista predeterminada de Google Maps es la de un mapa tradicional calle. Se puede visualizar un mapa introduciendo una dirección de calle, intersección de la calle (en la mayoría de grandes ciudades, ciudad o estado nombre, código postal, o la latitud y los números de longitud. Este tipo de cartografía básica es bastante fácil de hacer.

Para mostrar los mapas, se dividen las imágenes en cuadrados de 256x256 píxeles, denominados 'tiles', que se van descargando asíncronamente, generando una vista fluida.

Más allá de mapas simples de calle, Google Maps ofrece una serie de vistas de mapa. Estos incluyen:

- **Vista de tráfico**, que muestra la información de tráfico en tiempo real sobre el mapa de la calle. (Verde para el tráfico normal, rojo para realmente mal tráfico, etc.
- **Vista por satélite**, que muestra una vista de pájaro de un lugar determinado. (Es una gran manera de ver la parte superior de su casa, o ver su vecindario como se ve desde un avión. Aunque es conocida como vista de satélite, muchas de estas imágenes son en realidad fotografías aéreas de alta resolución tomadas desde aviones.
- **Street view**, que muestra las fotos a nivel de calle sobre la ubicación del mapa.

También hay una vista de Google Earth, que ofrece una versión en 3D la vista satelital tradicional que te permite volar a través y alrededor de la ubicación de un mapa.

Google Maps también permite visualizar mapas superpuestos con todo tipo de información útil. Por ejemplo, puede mostrar fotos y videos tomadas cerca de la ubicación del mapa, cámaras web en las cercanías, y las entradas de Wikipedia sobre ubicaciones cercanas.

Además, Google Maps ofrece información sobre navegaciones de punto a punto de en coche, caminando o en transporte público Sólo se tiene que introducir una ubicación de inicio y fin, y Google calcula la mejor manera de llegar del punto A al punto B y muestra instrucciones giro a giro y un mapa adjunto. No todas las funciones de Google Maps están disponibles para todas las ubicaciones.

Por si todo esto no fuera suficiente, Google ofrece una API para Google Maps (Application Programming Interface). Esta API permite a los desarrolladores integrar Google Maps en sus propios sitios web y dispositivos móviles, con sus propios datos. Este es el camino a seguir si se quiere desarrollar una aplicación basada en localización, tecnología que se ha utilizado en este proyecto fin de carrera.

Google Maps se encuentra integrado por defecto en las librerías de Android, por lo que es muy sencillo integrarlo en una aplicación. En un principio, se incluyó la versión 1 de Google Maps para Android, que es la que se ha utilizado para desarrollar la aplicación de "Contacta!", pero actualmente ya se dispone de la versión 2, en la que se han añadido las siguientes funcionalidades:

- Mapas en 3D, con posibilidad ver el mapa en perspectiva.
- Mapas en interiores de edificios.
- Se ha mejorado la caché de mapas, por lo que se quedarán almacenados en modo offline.
- Los 'tiles' utilizan vectores para representarse, lo que provoca que se disminuya el ancho de banda generado para descargarlos y aparecen antes en la aplicación.

Su aspecto es el siguiente:

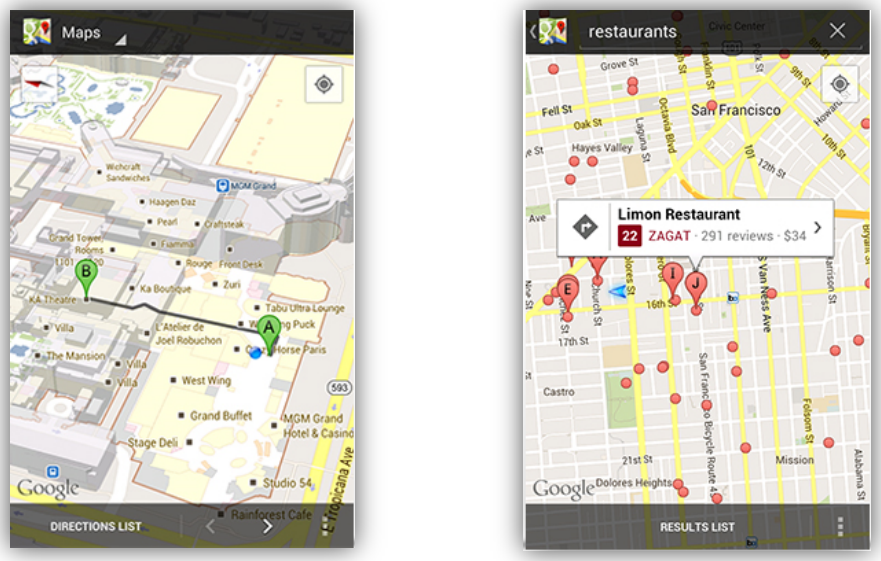


Figura 2.18: Aspecto de Google Maps para Android v2

## 2.12 Realidad Aumentada

La Realidad Aumentada (AR) [12, 13] es una variación de un Entorno Virtual (VE), o Realidad Virtual (VR) como es más comúnmente llamado. Las tecnologías de Realidad Virtual sumergen completamente a un usuario dentro de un entorno sintético y mientras se está inmerso, el usuario no puede ver la realidad que le rodea. Por el contrario, la Realidad Aumentada toma la información generada digitalmente, ya sean imágenes, audio, video, y toque o sensaciones hápticas, y la superpone en un entorno en tiempo real. La Realidad Aumentada técnicamente se puede utilizar para mejorar los cinco sentidos, pero su uso más común de hoy en día es visual. A diferencia de la Realidad Virtual, la Realidad Aumentada permite al usuario ver el mundo real, con objetos virtuales superpuestos o compuestos con el mundo real. Por lo tanto, AR suplementa la realidad, en lugar de sustituir por completo como se muestra en la Figura 2.19. La Realidad Aumentada se puede considerar como la mezcla o el "punto medio" entre lo completamente sintético y lo completamente real.

Figura 2.19: Un gráfico o modelo 3D se superpone sobre el objeto del mundo real mediante el uso de un smartphone

Uno de los ejemplos más fáciles es un heads-up display, HUD, utilizado por los pilotos de combate. Es probable que se hayan visto ejemplos de HUD en las películas o documentales de televisión. Un HUD da al piloto una superposición digital que muestra un horizonte artificial, la altitud digital, la velocidad digital, y otra información, mientras mira por la ventana de la cabina, como se muestra en la Figura 2.20.


Figura 2.20: HUD de un caza

Sobre la base de la definición básica y la descripción de las capacidades de AR permiten ampliar un poco más y resumir las tres características que deben estar presentes para la verdadera Realidad Aumentada:

1. AR combina información real y virtual.

2. AR es interactivo en tiempo real.

3. AR opera y se utiliza en un entorno 3D.

La Realidad Aumentada realmente permite presentar visualmente la información que el usuario no sería capaz de detectar. Así como hay millones de bits de información que se transmiten a nuestro alrededor en este momento en alguna frecuencia inalámbrica, estando la gente totalmente inconsciente de que los teléfonos móviles, tablets y portátiles son los que permiten canalizar eficazmente la información. La Realidad Aumentada, al igual que otras interfaces gráficas, da la posibilidad de llevar información útil en el espectro visual en tiempo real en cualquier lugar. No es sólo una tecnología, es la combinación de varias tecnologías que trabajan juntas para llevar la información digital en la percepción visual. AR es una muy convincente, casi interminable, colección de experiencias asistidas por tecnología que ayuda a crear la Web y aplicaciones en tiempo real.

Como Gene Becker de Lightning Laboratories dice, la realidad aumentada es:

- Una tecnología.
- Un campo de investigación.
- Una visión de la computación en el futuro.
- Una industria comercial emergente.
- Un nuevo medio de expresión creativa.

### 2.12.1 Usos actuales

A pesar de ser un campo relativamente nuevo, hay suficientes aplicaciones de AR disponibles, sobre todo para smartphones, para establecer categorías. Hay cientos de aplicaciones que utilizan AR que están destinadas para que las utilicen los usuarios de nivel medio. Los hay de muchos tipos, por ejemplo, juegos, mapas y aplicaciones de navegación. Por lo general utilizan el acelerómetro y el GPS para obtener la ubicación y el estado físico del dispositivo.

Hoy en día, se utilizan aplicaciones de Realidad Aumentada, con capacidad de desarrollarse en Android, en los siguientes campos:

- **Militar:** desde gafas con información del terreno, reconocimiento de imagen, etc. hasta simuladores para ayudar en los entrenamientos.
- **Vehículos:** navegadores GPS y ordenador de abordo proyectados en el cristal.
- **Medicina:** cirugías habilitadas con AR para reducir el riesgo de error.
- **Juicios:** AR para realizar juicios virtuales y así ahorrar dinero.

- **Turismo:** guías para ciudades.
- **Otros campos:** arquitectura, cine, entretenimiento, educación, arte, traducción, predicción del tiempo, televisión, astronomía…

### 2.12.2 Usos futuros

Como se analizó en la sección anterior, la Realidad Aumentada es bien conocida y tiene bastantes aplicaciones disponibles para que sea digna de mención. Sin embargo, existen algunos usos sorprendentes de la tecnología que no se puede implementar en este momento debido a las limitaciones de hardware y algoritmos.

- **Experiencias virtuales:** en el futuro, la AR podría ser usada para crear experiencias virtuales, en la que se montara un dispositivo en la cabeza del usuario que le trasladara a un lugar completamente distinto.
- **Hologramas:** AR permite al usuario tener una visión directa o indirecta en vivo del mundo, lo que podría permitir a los usuarios tener hologramas en frente de ellos. Estos hologramas podrían ser interactivos o meramente descriptivo.
- **Video conferencias:** la Realidad Aumentada podría permitir que varias personas aparecieran en la misma sala de conferencias, si un canal de video de una sala de conferencias les transmite. Las personas pueden usar una cámara web para "aparecer" en los asientos la sala, junto con los demás. Esto podría crear un entorno de colaboración, aunque los colaboradores estuvieran a miles de kilómetros de distancia.
- **Películas:** se podría utilizar para reproducir películas enteras. El teatro podría ser reemplazado por el fondo de la película o el teatro podría ser reemplazado solamente por los actores. En la primera forma, los actores podrían ser “aumentados” en el fondo y en el segundo método el fondo podría ser “aumentado” detrás de los actores. Se podrían realizar películas más realistas y divertidas, mientras se mantiene el coste del rodaje.
- **Otros:** simulaciones imposibles, control de gestos.

La Realidad Aumentada ha recorrido un largo camino desde sus inicios y tiene un largo camino por recorrer. Los requisitos básicos de una cámara, GPS, acelerómetro y brújula se cumplen en casi todos los dispositivos Android en el mercado. Aunque existen aplicaciones que utilizan la tecnología de AR para la plataforma Android, son pocas en número en comparación con los otros tipos de aplicaciones. Es un buen momento para entrar en la plataforma Android, haciendo aplicaciones de AR porque la competencia es suficiente para impulsar el interés del usuario a utilizar estas aplicaciones, pero no lo suficientemente fuerte para que lo lleve a la quiebra todavía. Teniendo en cuenta las relativamente pocas aplicaciones de AR en el mercado, también hay una buena probabilidad de que si se realiza una buena aplicación no tendrá más de 3-5 aplicaciones de la competencia, lo que da una gran ventaja.

### 2.12.3 Librería de Realidad Aumentada Mixare

Para realizar este proyecto, se ha utilizado una librería de Realidad Aumentada llamada Mixare (mix Augmented Reality Engine) [14], que es un navegador de AR de código libre, abierto, que se publica bajo la licencia GPLv3. Mixare está disponible para Android y para el iPhone 3G y superiores. Funciona como una aplicación completamente autónoma y está disponible también para el desarrollo de implementaciones propias.

Mixare tiene múltiples formas de integrarse en una aplicación:

1. Mixare es una aplicación autónoma, que muestra los puntos de interés de Wikipedia de los alrededores.

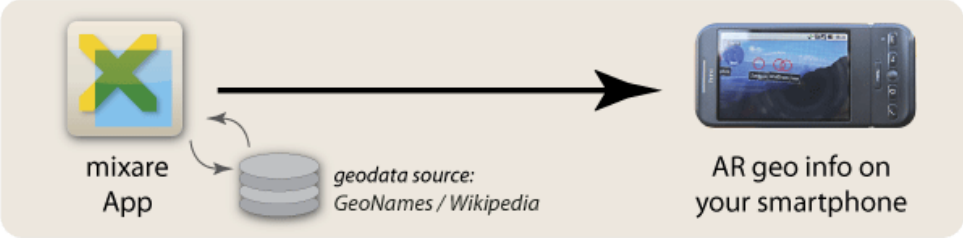


Figura 2.21: Uso de Mixare como aplicación autónoma

2. Mixare puede accederse a través de un link en una página HTML, donde la fuente de los datos se transmite a la aplicación.

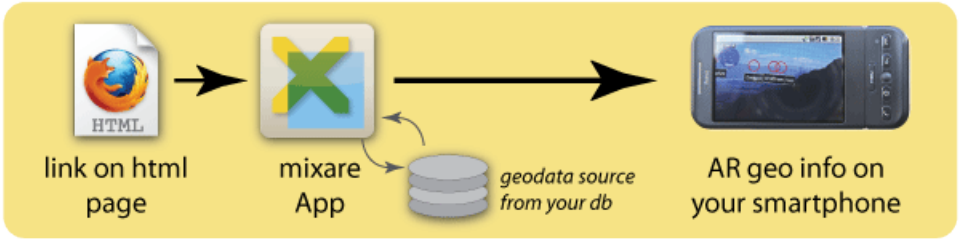


Figura 2.22: Uso de Mixare a través de una web

3. Mixare puede accederse a través de otra aplicación, donde se transmite la fuente de datos.

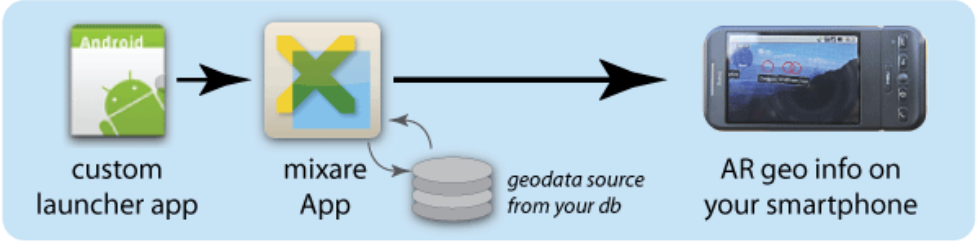


Figura 2.23: Uso de Mixare a través de otra aplicación

**4.** Mixare es libremente ampliable y se puede incluso integrar en una aplicación propia.

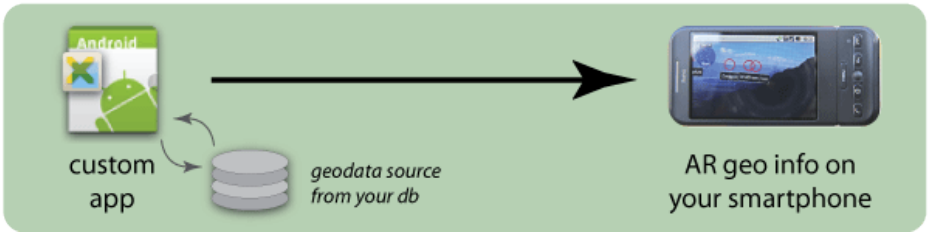


Figura 2.24: Integración de Mixare en una aplicación

La última opción es la necesaria para realizar la aplicación Android de este proyecto.

# Capítulo 3: Requisitos

*Este capítulo se centrará en una parte muy importante del desarrollo software: la captura y el análisis de requisitos. Los requisitos son una característica que un sistema software necesita para tener valor y utilidad para un usuario. Para esto, es necesario hacer un estudio detallado de los Casos de Uso, con ayuda del Lenguaje Unificado de Modelado (UML), cuyo objetivo es estructurar los requisitos del sistema en función de objetivos de usuario final.*

*Los Casos de Uso dan la posibilidad de representar de manera gráfica las especificaciones de un sistema, proporcionando un resultado observable y relevante para el usuario. Este capítulo tiene como finalidad la captura completa de los requisitos del sistema software para poder definir posteriormente la fase de diseño e implementación.*

# 3.1 Casos de Uso

El presente proyecto tiene como objetivo el desarrollo de la aplicación para Android "Contacta!", consistente en una red social formada por los contactos que contiene la agenda del móvil, en la que se podrá ver dónde se encuentran (tanto en el mapa como por realidad aumentada), el estado, llamar, enviar SMS y chatear con ellos.

A continuación se introducirán los actores y se procederá, mediante un esquema definido, a detallar los Casos de Uso que ilustran los requisitos y el funcionamiento del sistema, desde el punto de vista del usuario, mostrando cómo reacciona éste a eventos que se producen en su ámbito o en él mismo.

## 3.1.1 Actores

En tabla 3.1 se muestran los actores primarios y secundarios del sistema.

| Papel | Descripción |
|---|---|
| Usuario | Usuario del sistema |
| Administrador | Administrador del sistema |
| Usuario no registrado | Usuario no registrado en el sistema |

Tabla 3.1: Actores del sistema

## 3.1.2 Diagrama de Casos de Uso

En la figura 3.1 se muestran los Casos de Uso de la aplicación mediante un diagrama, en el que podemos observar las acciones que pueden realizar los actores del sistema.

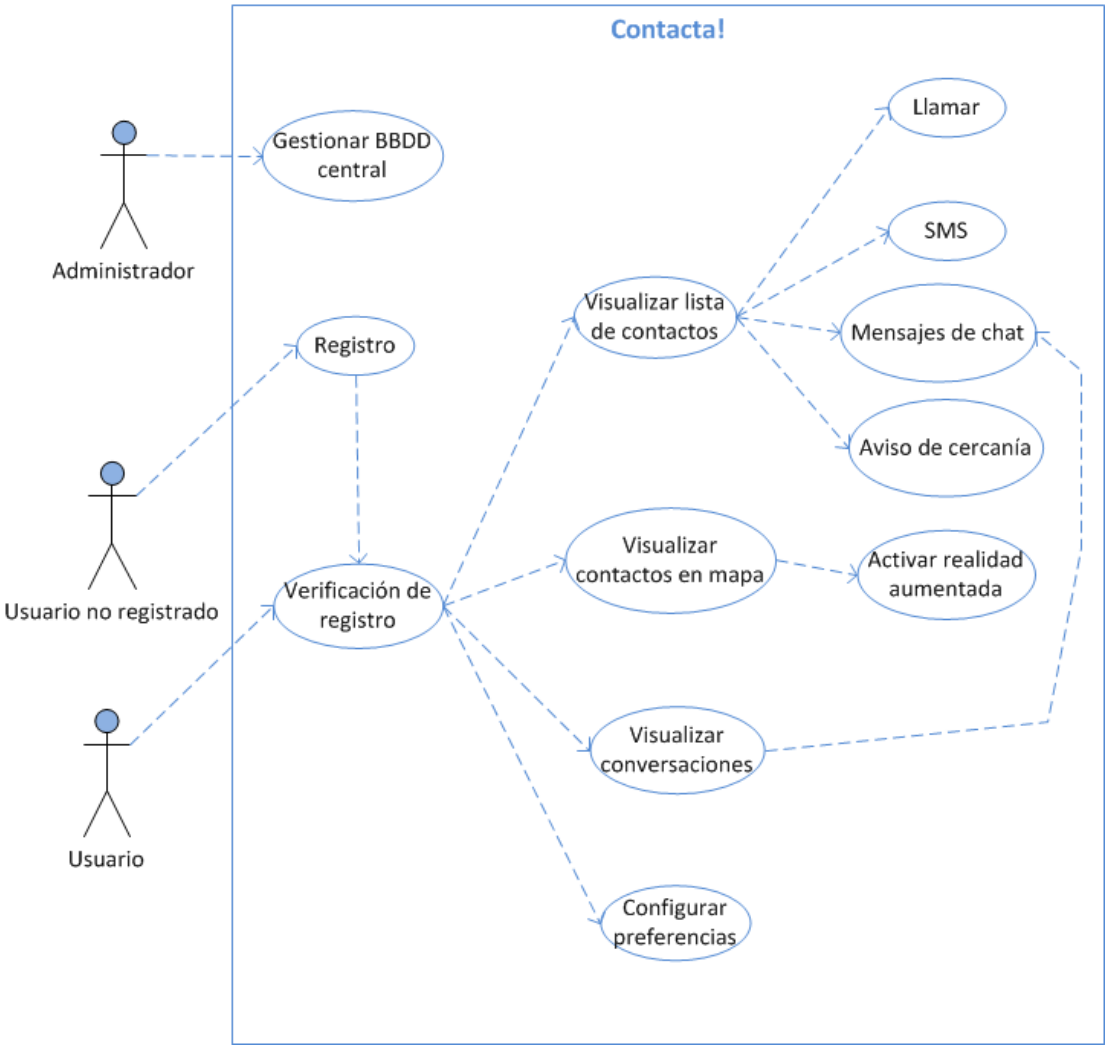


Figura 3.1: Diagrama de Casos de Uso

### 3.1.3 Detalle de acciones posibles

Una vez definidos los Casos de Uso, se detallarán a continuación en las siguientes tablas (ver Tabla 3.2 hasta 3.13).

- **Caso de Uso 1:** Gestión de la Base de Datos Central

| | |
|---|---|
| **Nombre** | Gestión de la Base de Datos Central |
| **Descripción** | Mantener y gestionar las tablas correspondientes a los datos de los usuarios de la aplicación “Contacta!” |
| **Actores** | Administrador |
| **Precondiciones** | Login en la BBDD |
| **Flujo Normal** | 1. El administrador accede a la BBDD PostgreSQL<br>2. Gestiona las tablas según sea necesario |
| **Excepciones** | Si no se hace login correctamente, no tendrá acceso a la BBDD |
| **Poscondiciones** | - |

Tabla 3.2: Caso de Uso 1

- **Caso de Uso 2:** Registro

| | |
|---|---|
| **Nombre** | Registro |
| **Descripción** | Se realiza un registro con el número de teléfono tanto en el móvil como en la base de datos del servidor |
| **Actores** | Usuario no registrado |
| **Precondiciones** | No existencia de un usuario con el mismo número de teléfono |
| **Flujo Normal** | 1. Se introduce el número de teléfono en el campo correspondiente<br>2. Se hace click en el botón de aceptar<br>3. Se comprueba que el usuario es único y se accede a la aplicación |
| **Excepciones** | Existe un registro anterior |
| **Poscondiciones** | Creación del usuario en las BBDD del teléfono y del servidor |

Tabla 3.3: Caso de Uso 2

- **Caso de Uso 3:** Verificación de registro

| | |
|---|---|
| **Nombre** | Verificación de registro |
| **Descripción** | Se realiza una verificación de que el usuario se ha registrado previamente |
| **Actores** | Usuario |
| **Precondiciones** | Existencia de un usuario registrado en la aplicación |
| **Flujo Normal** | 1. Se abre la aplicación<br>2. Mediante los datos guardados en la aplicación, se comprueba que hay un usuario registrado |
| **Excepciones** | - |
| **Poscondiciones** | - |

Tabla 3.4: Caso de Uso 3

- **Caso de Uso 4:** Visualizar lista de contactos

| | |
|---|---|
| **Nombre** | Visualizar lista de contactos |
| **Descripción** | Se muestra una lista de dos secciones en las que aparecen los contactos del teléfono del usuario que tengan instalado “Contacta!” y los que no |
| **Actores** | Usuario |
| **Precondiciones** | Estar registrado y tener contactos en el móvil |
| **Flujo Normal** | 1. La pestaña Contactos es la primera que aparece al abrir la aplicación, mostrando la lista |
| **Excepciones** | - |
| **Poscondiciones** | - |

Tabla 3.5: Caso de Uso 4

- **Caso de Uso 5:** Llamar

| **Nombre** | Llamar |
|---|---|
| **Descripción** | Se llama a un contacto del usuario |
| **Actores** | Usuario |
| **Precondiciones** | Estar registrado y tener contactos en el móvil |
| **Flujo Normal** | 1. En la pestaña Contactos, se deja pulsado a quien se quiera llamar y aparece un menú contextual<br>2. Hacer click en el botón “Llamar” del menú contextual |
| **Excepciones** | No se dispone de capacidad de realizar una llamada |
| **Poscondiciones** | - |

Tabla 3.6: Caso de Uso 5

- **Caso de Uso 6:** SMS

| **Nombre** | SMS |
|---|---|
| **Descripción** | Se envía un SMS a un contacto del usuario |
| **Actores** | Usuario |
| **Precondiciones** | Estar registrado y tener contactos en el móvil |
| **Flujo Normal** | 1. En la pestaña Contactos, se deja pulsado a quien se quiera enviar un SMS y aparece un menú contextual<br>2. Hacer click en el botón “SMS” del menú contextual |
| **Excepciones** | No se dispone de capacidad de enviar SMS |
| **Poscondiciones** | - |

Tabla 3.7: Caso de Uso 6

- **Caso de Uso 7:** Mensajes de chat

| **Nombre** | Mensajes de chat |
|---|---|
| **Descripción** | Se establece una conversación de Chat con un contacto del usuario |
| **Actores** | Usuario |
| **Precondiciones** | Estar registrado y tener algún contacto que disponga de la aplicación |
| **Flujo Normal** | 1. En la pestaña Contactos, se deja pulsado a un contacto que disponga "Contacta!" al que se quiera enviar un mensaje de Chat y aparece un menú contextual<br>2. Hacer click en el botón "Chat" del menú contextual<br>3. Se abre una ventana con la conversación que se tenga con dicho contacto y aparece una caja de texto donde se escribe el mensaje<br>4. Una vez escrito, se pulsa el botón enviar |
| **Excepciones** | - |
| **Poscondiciones** | - |

Tabla 3.8: Caso de Uso 7

- **Caso de Uso 8:** Aviso de cercanía

| **Nombre** | Aviso de cercanía |
|---|---|
| **Descripción** | Se establece un aviso de cercanía para un contacto, con el que se emitirá una notificación cuando éste se encuentre dentro de un radio establecido |
| **Actores** | Usuario |
| **Precondiciones** | Estar registrado y tener algún contacto que disponga de la aplicación con la localización activada |
| **Flujo Normal** | 1. En la pestaña Contactos, se marca la casilla de un contacto que disponga de "Contacta!" con la localización activada |
| **Excepciones** | - |
| **Poscondiciones** | - |

Tabla 3.9: Caso de Uso 8

- **Caso de Uso 9:** Visualizar contactos en mapa

| | |
|---|---|
| **Nombre** | Visualizar contactos en mapa |
| **Descripción** | Se muestra un mapa en el que se sitúan los contactos con su foto |
| **Actores** | Usuario |
| **Precondiciones** | Estar registrado y tener algún contacto que disponga de la aplicación con la localización activada |
| **Flujo Normal** | 1. Se selecciona la pestaña Mapa. |
| **Excepciones** | - |
| **Poscondiciones** | - |

Tabla 3.10: Caso de Uso 9

- **Caso de Uso 10:** Activar realidad aumentada

| | |
|---|---|
| **Nombre** | Activar realidad aumentada |
| **Descripción** | Se muestra la cámara del teléfono, en la que se dibujan los contactos según su posición |
| **Actores** | Usuario |
| **Precondiciones** | Estar registrado, tener cámara y algún contacto que disponga de la aplicación con la localización activada |
| **Flujo Normal** | 1. Una vez se está en la pestaña Mapa, se pulsa el botón menú del teléfono, y abre un menú contextual<br>2. Se hace click en el botón “Cámara” |
| **Excepciones** | - |
| **Poscondiciones** | - |

Tabla 3.11: Caso de Uso 10

- **Caso de Uso 11:** Visualizar conversaciones

| | |
|---|---|
| **Nombre** | Visualizar conversaciones |
| **Descripción** | Se muestra la una lista con las conversaciones establecidas con los contactos |
| **Actores** | Usuario |
| **Precondiciones** | Estar registrado, tener alguna conversación con un contacto que disponga de la aplicación |
| **Flujo Normal** | 1. Se selecciona la pestaña “Chat”<br>2. Aparece una lista con las conversaciones y se hace click en una<br>3. Aparece una ventana con los mensajes intercambiados con el contacto |
| **Excepciones** | - |
| **Poscondiciones** | - |

Tabla 3.12: Caso de Uso 11

- **Caso de Uso 12:** Configurar preferencias

| | |
|---|---|
| **Nombre** | Configurar preferencias |
| **Descripción** | Se muestran las opciones posibles a configurar: aplicación al inicio del sistema, mostrar ubicación, frecuencia de actualización, radio de proximidad y estado |
| **Actores** | Usuario |
| **Precondiciones** | Estar registrado |
| **Flujo Normal** | 1. Una vez abierta la aplicación, en cualquiera de las pestañas principales, se hace click en el botón menú del teléfono<br>2. Se pulsa el botón “Preferencias”<br>3. Aparece una lista con las opciones a modificar |
| **Excepciones** | - |
| **Poscondiciones** | - |

Tabla 3.13: Caso de Uso 12

## 3.2 Captura y análisis de requisitos

Tras haber descrito los Casos de Uso anteriores, ahora se pueden definir las especificaciones generales que debe cumplir el sistema y así se tendrá una base con la que comenzar a implementar la arquitectura de la aplicación. Para ello es necesario capturar los requisitos, que son una capacidad, característica o factor de calidad que un sistema software necesita para tener valor y utilidad para un usuario.

En la Tabla 3.14 aparecen los requisitos capturados:

| Identificador | Descripción | Tipo de requisito | Prioridad | Trazabilidad |
|---|---|---|---|---|
| 1 | Pantalla de registro en la aplicación, para que el usuario pueda introducir su número de teléfono | Funcional | Obligatoria | Caso de Uso 2 |
| 2 | Pantalla de visualización de lista de contactos, tanto los que dispongan de “Contacta!” como los que no | Funcional | Obligatoria | Caso de Uso 3 |
| 3 | Pantalla de visualización de la posición tanto en mapa como en realidad aumentada de los contactos que tengan la posición visible | Funcional | Obligatoria | Caso de Uso 8 y 9 |
| 4 | Pantalla de visualización de las conversaciones establecidas con contactos | Funcional | Obligatoria | Caso de Uso 10 |
| 5 | Pantalla de control de las preferencias de la aplicación | Funcional | Obligatoria | Caso de Uso 11 |
| 6 | Establecer un acceso a la base de datos central para el administrador, para que pueda crear y modificar las tablas pertinentes | Funcional | Obligatoria | Caso de Uso 1 |

Tabla 3.14: Requisitos capturados

# Capítulo 4: Arquitectura

*En este capítulo se va a desarrollar la arquitectura de la aplicación, respetando los requisitos capturados y analizados anteriormente. En un principio, se mostrará la arquitectura general del sistema, que dará una visión global de sus componentes y cómo se interconectan. Luego se explicará la parte del servidor y del cliente, detallando sus componentes principales.*

## 4.1 Arquitectura general del sistema

A continuación se muestra un diagrama que explica a grandes rasgos la arquitectura empleada en "Contacta!", que es cliente-servidor, ya que es fácilmente distribuible y escalable, permitiendo distribuir el modelo de datos entre cliente y servidor. En él se puede observar las interacciones entre la parte de cliente, servidor y base de datos.

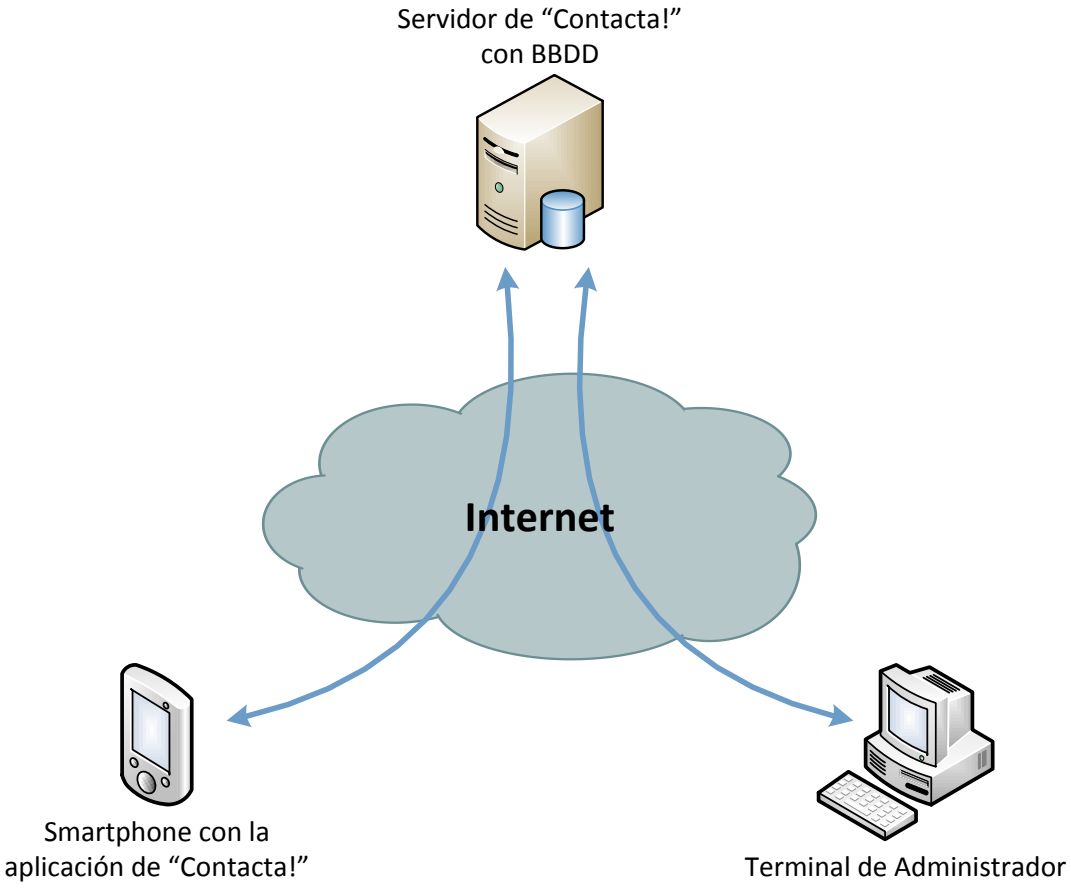


Figura 4.1: Arquitectura general del sistema

## 4.2 Arquitectura del servidor

El servidor se encarga de guardar en una base de datos todos los datos de los usuarios de la aplicación. Dichos usuarios transmiten su información a través de la aplicación cliente y hacen peticiones para recibir la de los demás. Para lograr este propósito, el servidor estará formado por una serie de componentes, que se ven en el siguiente diagrama:

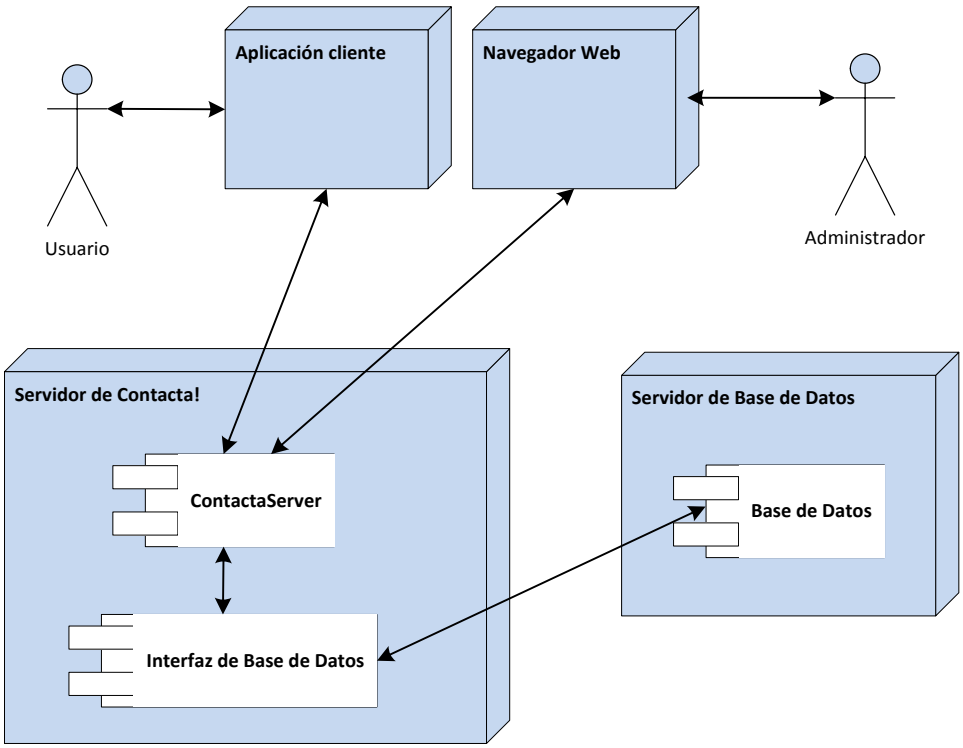


Figura 4.2: Diagrama de componentes del servidor

Se ha decidido realizar el servidor en Java mediante un Servlet, usando PostgreSQL para la base de datos. Como se puede observar en el diagrama, hay tres componentes fundamentales:

- **ContactaServer:** es el Servlet, se encarga de recibir las peticiones de los usuarios y administrador y mostrar información.
- **Interfaz de Base de Datos:** tiene como objetivo conectar el Servlet a la Base de Datos, para que pueda leer y escribir en ella. Se utiliza el driver JDBC de la clase `org.postgresql.Driver`.
- **Base de Datos:** es el sistema de almacenamiento de datos de los usuarios, realizado con la tecnología PostgreSQL.

El servidor le transmite los datos al cliente en formato JSON, sin embargo, el cliente envía datos al servidor mediante comandos GET de HTML.

Para ver en más profundidad el contenido de cada componente, a continuación se muestra un diagrama de clase:

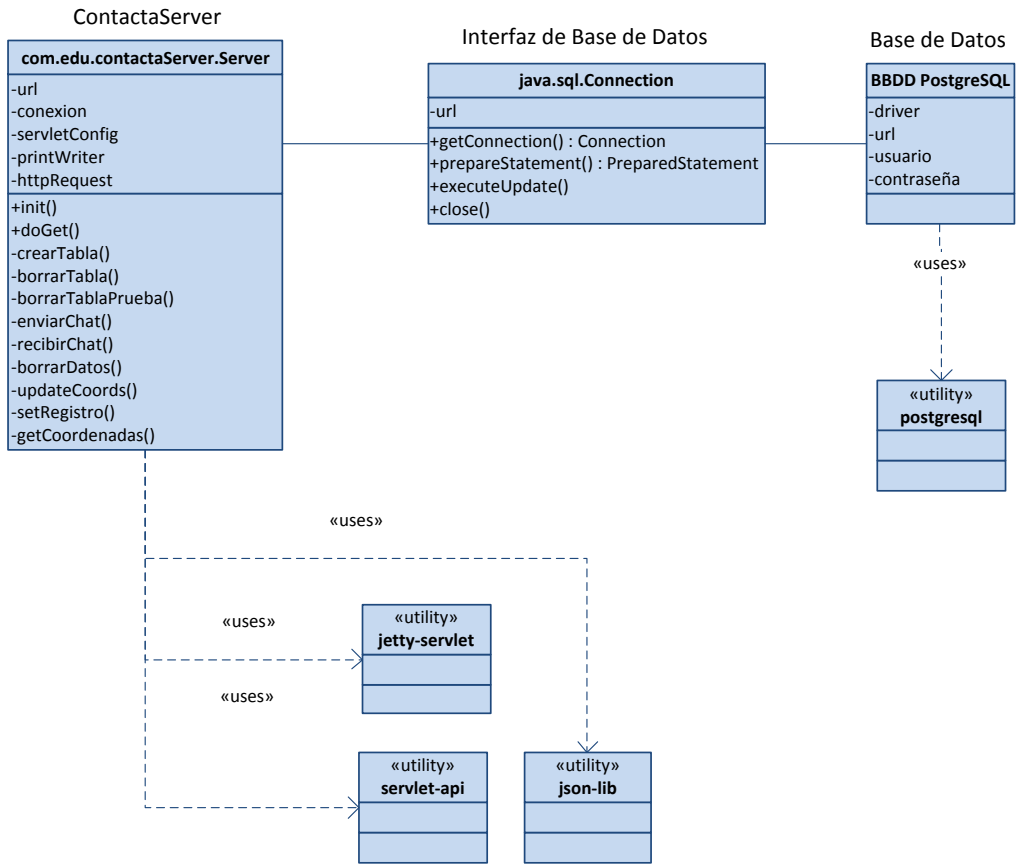


Figura 4.3: Diagrama de clase del servidor

En los siguientes apartados, se explicará en detalle cada clase junto con sus atributos, operaciones y relaciones.

### 4.2.1 ContactaServer

- **Paquete:** `com.edu.ContactaServer.Server`
- **Atributos:** propiedades características compartidas por todos los objetos de una clase. Tiene los siguientes:

    - **url:** String que contiene la información de la dirección de la base de datos, el usuario y contraseña para acceder y el driver a utilizar.
    - **conexion:** objeto de la clase `java.sql.Connection` que se utiliza para realizar las conexiones con la base de datos.
    - **servletConfig:** objeto `javax.servlet.ServletConfig` en el que se almacena la configuración inicial del Servlet cuando se crea.
    - **printWriter:** objeto `java.io.PrintWriter` encargado de hacer mostrar las respuestas del Servlet.
    - **httpRequest:** objeto javax.servlet.http.HttpServletRequest que obtiene los parámetros indicados por el cliente.
- **Operaciones:** son los servicios básicos ofrecidos por los objetos de una clase. Su implementación se denomina método. Se dispone de los siguientes:

    - **init():** inicia el Servlet con una configuración específica.

- **doGet():** obtiene los parámetros enviados por el cliente en las llamadas GET y según los que sean llama a un método u otro.
- **crearTabla():** crea en la base de datos la tabla 'prueba' con los siguientes campos: numero, latitud, longitud, altitud, precision, estado, ubicar.
- **borrarTabla():** borra la tabla de la base de datos cuyo nombre se le pase como parámetro.
- **borrarTablaPrueba():** borra la tabla 'prueba' de la base de datos.
- **enviarChat():** obtiene del cliente el número origen, destino y mensaje; crea en la base de datos una tabla con el nombre 'Tabla_DESTINO' (si no estuviera ya creada) con las columnas origen, hora y mensaje, e inserta una fila con los datos obtenidos del cliente más la hora actual del servidor.
- **recibirChat():** recibe del cliente un número destino y una hora y hace una consulta a la base de datos y le devuelve al cliente todas las entradas de la tabla 'Tabla_DESTINO' a partir de la hora dada.
- **borrarDatos():** elimina la entrada de la tabla 'prueba' cuyo número coincida con el que proporciona la aplicación cliente.
- **updateCoords():** actualiza los datos de latitud, longitud, altitud, precision, estado y ubicar, proporcionados por el cliente, para un número determinado.
- **setRegistro():** registra un nuevo usuario en la base de datos, creando una entrada en la tabla 'prueba' con los datos correspondientes obtenidos del cliente.
- **getCoordenadas():** devuelve todos los datos y coordenadas de la tabla 'prueba'.

- **Dependencias:** depende de tres componentes:
  - **jetty-servlet:** proporciona un servidor web capaz de soportar servlets.
  - **servlet-api:** contiene una serie de clases e interfaces que describen y definen los contratos entre una clase servlet y el entorno de ejecución.
  - **json-lib:** librería que convierte los datos que se envían al cliente al formato JSON.

### 4.2.2 Interfaz de Base de Datos

- **Paquete:** `com.sql.Connection`
- **Atributos:**
  - **url:** Parámetro que guarda la dirección para acceder a la base de datos, obtenida de ContactaServer.
- **Operaciones:**
  - **getConnection():** crea un objeto Connection a partir de una URL dada y se conecta a la base de datos.
  - **prepareStatement():** crea un objeto PreparedStatement para enviar sentencias SQL parametrizadas a la base de datos.
  - **executeUpdate():** ejecuta una sentencia SQL en un objeto PreparedStatement.
  - **close():** cierra la conexión establecida con la base de datos.

### 4.2.3 Base de Datos

- **Atributos:**
  - **driver:** tipo de driver que utiliza para comunicarse con el Servlet, en este caso es `jdbc:postgresql`.
  - **url:** dirección donde está alojada la base de datos.

- **Usuario y contraseña:** credenciales para poder acceder a la base de datos.

- **Dependencias:**
  - **Postgresql**

## 4.3 Arquitectura del cliente

El cliente es la aplicación Android, que se instalará en el terminal de usuario. A través de una conexión a internet, el cliente se conectará con el servidor y podrá interactuar con él; interacciona con una base de datos en local, con basada en SQLite, en la que guardará los datos que reciba del servidor, para un funcionamiento más rápido. El cliente dispone de numerosos componentes que permiten cumplir las especificaciones generales del sistema, como se ve en el diagrama a continuación:

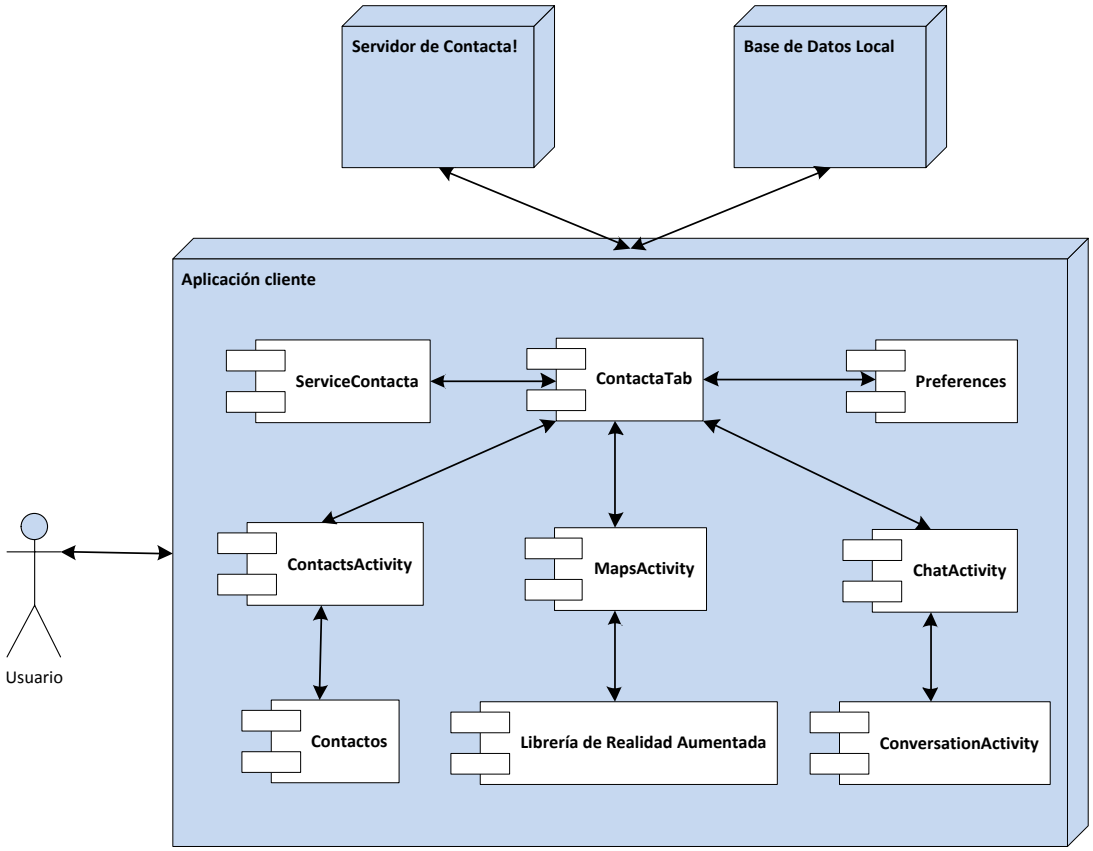


Figura 4.4: Diagrama de componentes de cliente

La aplicación de Android tiene 9 componentes principales:

- **Preferences:** Actividad de preferencias que permite personalizar las opciones de la aplicación.
- **ServiceContacta:** Servicio que se encarga de obtener la ubicación del dispositivo y los datos del servidor periódicamente, actualizando el campo del servidor que corresponda al dispositivo cuando haya cambiado algún parámetro.

- **ContactaTab:** Es una Actividad cuya finalidad es mostrar la vista por pestañas, que son las siguientes: ContactsActivity, MapsActivity y ChatActivity. Establece como vista principal ContactsActivity.
- **ContactsActivity:** Es la vista principal de la aplicación. En ella aparecen los contactos que tengan la aplicación instalada y estén añadidos a la agenda y también, por separado, todos los contactos de la agenda**.** Se encarga de pedir registro si el usuario no se ha registrado.
- **Contactos:** Es el modelo de datos para los contactos.
- **MapsActivity:** Vista del mapa. En ella se dibujan los contactos que tengan la aplicación instalada y estén en la agenda, y tengan la ubicación disponible. También se pueden ver a través de la Realidad Aumentada.
- **Librería de Realidad Aumentada:** Es la librería utilizada para renderizar en la vista de la cámara los contactos que compartan su posición.
- **ChatActivity:** Muestra en una lista las conversaciones de chat establecidas.
- **ConversationActivity:** Abre un diálogo en el que se puede establecer una conversación de chat con un contacto.

Con el siguiente diagrama, se pretende explicar en profundidad los componentes y sus relaciones:

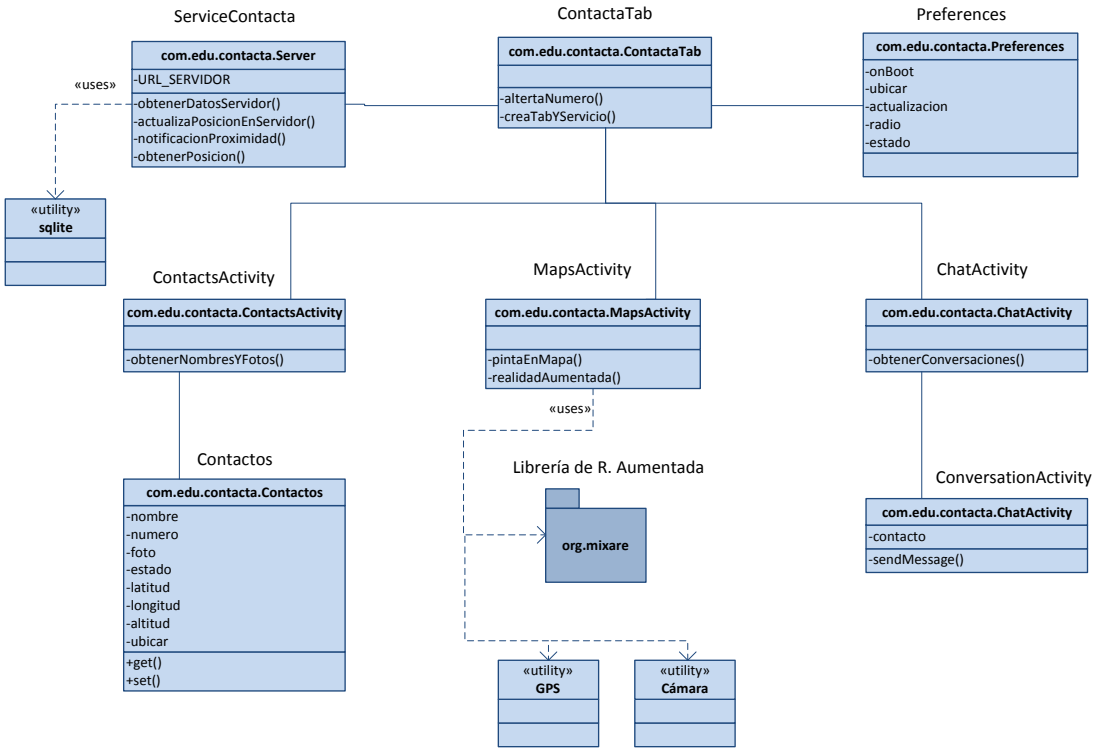


Figura 4.5: Diagrama de clases del cliente

Para una mejor comprensión del funcionamiento del cliente, a continuación se muestran unos diagramas de secuencia en los que se pueden ver los mensajes intercambiados entre los componentes principales y su orden temporal.

- **Diagrama de secuencia de cliente 1:** Visualización de contactos en ContactsActivity y actualización de la posición en el servidor.

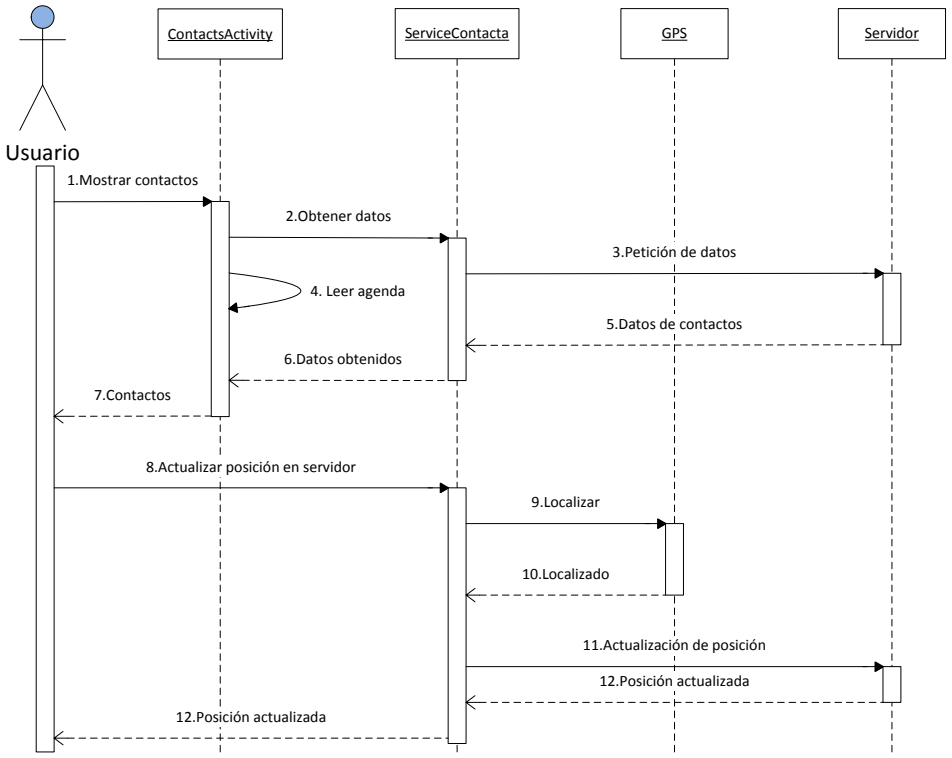


Figura 4.6: Diagrama de secuencias de cliente 1

- **Diagrama de secuencia de cliente 2:** Visualización de contactos en MapsActivity y mediante Realidad Aumentada.

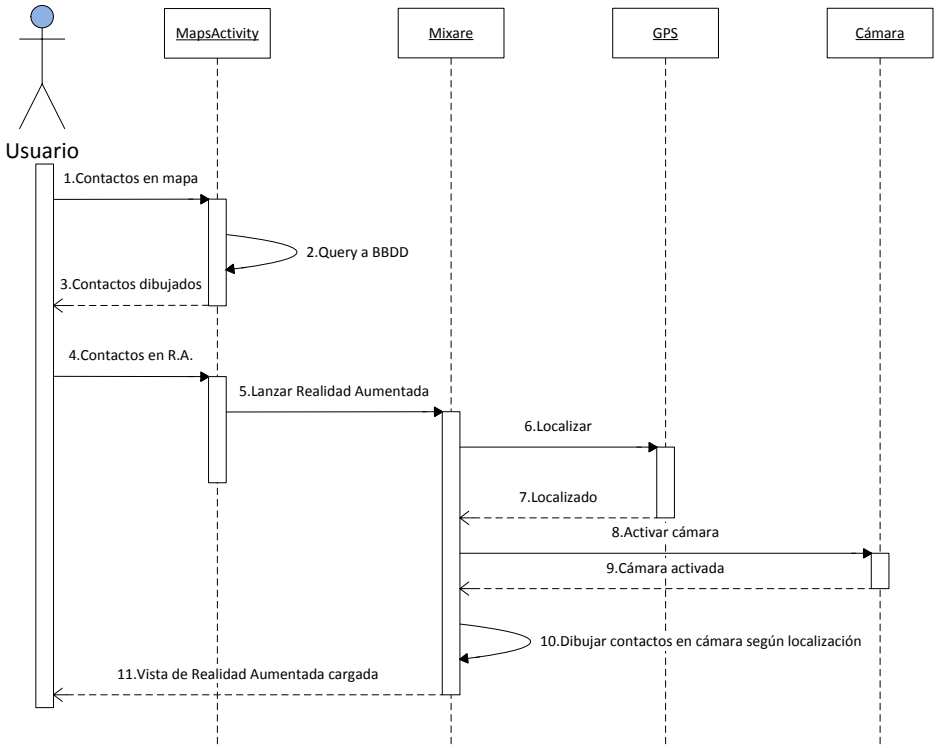


Figura 4.7: Diagrama de secuencias de cliente 2

- **Diagrama de secuencia de cliente 3:** Visualización de conversaciones en ChatActivity y envío de un mensaje de chat.

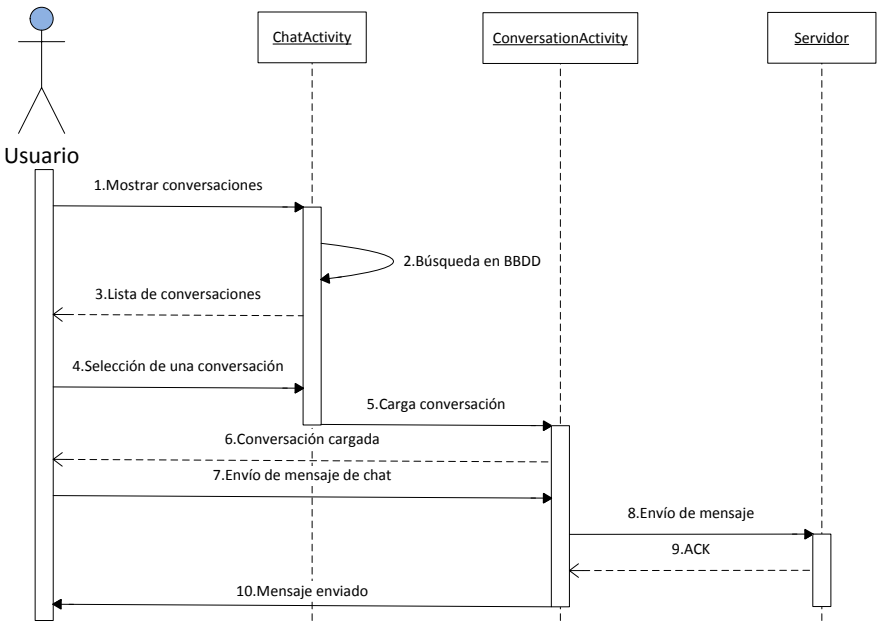


Figura 4.8: Diagrama de secuencias de cliente 3

A continuación, se explicará cada clase:

### 4.3.1 Preferences

- **Paquete:** com.edu.contacta.Preferences
- **Atributos:**
  - **onBoot:** Parámetro que indica si la aplicación se inicia al arrancar el terminal.
  - **ubicar:** Permite compartir la ubicación con los demás contactos.
  - **actualizacion:** Establece la frecuencia con la que actualizar los datos.
  - **radio:** Distancia mínima a un contacto para recibir una notificación de proximidad.
  - **estado:** Campo en el que se establece una frase identificando el estado del usuario.

La clase Preferences es una actividad que hereda de PreferenceActivity. En Android se utiliza para mostrarle una ventana al usuario en la que pueda cambiar las opciones de la aplicación. Para definirla hay que crear un fichero XML en el directorio "res/xml", en el que se especifican los parámetros a cambiar. El elemento padre es PreferenceScreen, que es la clase donde señala la PreferenceActivity. Se pueden crear diferentes secciones mediante elementos PreferenceCategory, dentro de las cuales van los elementos que el usuario puede modificar, y los más comunes suelen ser CheckBoxPreference (para definir propiedades booleanas), EditTextPreference (para introducir texto), y ListPreference (lista de valores predefinidos para seleccionar).

A los elementos se les puede especificar un nombre, un resumen, un valor por defecto, una clave para poderlos referenciar desde el código y una lista de valores para las ListPreferences. El fichero que define las preferencias podría quedar de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
   <PreferenceCategory>
      <CheckBoxPreference
         android:title="string@titleCheckBox"
         android:defaultValue="false"
         android:summary="@string/summaryCheckBox"
         android:key="enInicio" />
      <ListPreference
         android:title="@string/titleList"
         android:summary="@string/summaryList"
         android:key="actualizacion"
         android:defaultValue="10000"
         android:entries="@array/listArray"
         android:entryValues="@array/listValues" />
      <EditTextPreference
         android:title="@string/nameEditText"
         android:summary="@string/summaryEditText"
         android:defaultValue=""
         android:key="editEstado" />
```

Estos campos almacenan sus valores mediante las preferencias compartidas de la aplicación (SharedPreferences), que es una forma de almacenamiento muy sencilla que ofrece Android para guardar pares de clave-valor y compartirlos con otros métodos y actividades. Aunque se cierre la aplicación o se reinicie el terminal, dichos valores permanecerán a salvo.

Para recuperarlos desde cualquier actividad, habría que hacer lo siguiente:

```
SharedPreferences settings =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
boolean valor = settings.getBoolean("clave", defValue);
```

### 4.3.2 ServiceContacta

- **Paquete:** com.edu.contacta.ServiceContacta
- **Atributos:**
  - **URL_SERVIDOR:** Dirección del servidor para hacer las consultas.
- **Operaciones:**
  - **obtenerDatosServidor():** se conecta al servidor y obtiene los datos correspondientes al usuario.
  - **actualizaPosicionEnServidor():** actualiza los datos del usuario en el servidor (campos número, latitud, longitud, altitud, precisión, estado y ubicar).
  - **notificacionProximidad():** genera una notificación indicando que hay un contacto dentro del radio establecido en las preferencias.
  - **obtenerPosicion():** obtiene la posición del terminal del usuario.
- **Dependencias:**
  - **sqlite:** base de datos en local donde guarda toda la información proporcionada por el servidor.

ServiceContacta es un servicio, un componente sin interfaz gráfica destinado a realizar tareas en segundo plano sin interaccionar con el usuario. Se ejecuta periódicamente según el tiempo que se indique en las preferencias y tiene tres funciones principales:

1. Descarga los datos en JSON correspondientes del servidor, los lee y los introduce en la base de datos sqlite local para que las actividades puedan acceder a ellos.
   Es muy sencillo introducir datos en la base de datos sqlite local: se crea una instancia de la base de datos, se crea un objeto ContentValues, que almacenará las parejas clave-valor y se inserta dicho objeto en la BBDD, como se puede ver a continuación:

```
SQLiteDatabase mDatabase = openOrCreateDatabase(DATABASE_NAME, MODE_PRIVATE,
null);
ContentValues values = new ContentValues();
values.put("clave", valor);
mDatabaseChat.beginTransaction();
mDatabase.insertOrThrow(TABLE_NAME, null, values);
mDatabaseChat.setTransactionSuccessful();
```

2. Localiza al usuario y envía su posición actualizada al servidor, periódicamente. Esto se realiza utilizando el GPS del dispositivo, si está activado, y si no mediante A-GPS, que obtiene la posición a través de los puntos de acceso WiFi que detecta alrededor y de las antenas de cobertura móvil. Esto se consigue de la siguiente manera:

```
LocationManager location = (LocationManager)
getSystemService(Context.LOCATION_SERVICE); //Crea un objeto Location,
encargado de obtener la ubicación
String best = location.getBestProvider(criteria, true); //Obtiene el método más
preciso de todos los posibles
location.requestLocationUpdates(best, updateRate, 0, trackListener);//Pide
localizaciones cada updateRate milisegundos, y las capturará
trackListener, un objeto de la clase LocationListener

//Una vez obtenida la localización, se llama al método onLocationChanged
del objeto trackListener:

public void onLocationChanged(Location location) {
   double lat = location.getLatitude();
   double lon = location.getLongitude();
```

3. Genera una notificación cuando un contacto se encuentra a una distancia menor de la proporcionada por el usuario en las preferencias. Para ello, es necesario leer de la base de datos qué contactos están marcados para recibir notificaciones de proximidad. Esto se consigue haciendo una query en la BBDD, cuyo resultado se almacena en un objeto de la clase `android.database.Cursor`, sobre el que se va iterando para leer su contenido. A continuación, en pseudocódigo se podrá ver cómo se consigue esta funcionalidad:

```
void notificacionProximidad() {

   Se inicializa la base de datos;
   Query a la base de datos para obtener los contactos;
   do{
      if(notificar == true){
         Calcula la distancia entre el usuario y el contacto;
         if(distancia < radio){
            Genera notificación para alertar al usuario;
         }
      }
   }while(haya más contactos);
```

Para calcular la distancia entre el usuario y el contacto, se utiliza el método `android.location.Location.distanceBetween`, que recibe como parámetro la latitud y longitud de cada uno y computa la distancia aproximada en metros del camino más corto entre los dos puntos. La distancia se define usando el elipsoide WGS84.

4. Genera una notificación cuando llega un nuevo mensaje de chat. Esto se consigue guardando el timestamp del último mensaje y pidiendo al servidor la nueva información a partir de ese momento en concreto.

### 4.3.3 ContactaTab

- **Paquete:** com.edu.contacta.ContactaTab
- **Operaciones:**
  - **alertaNumero():** verifica si el usuario se ha registrado, y si no lo ha hecho, le pide su número de teléfono para registrarlo en el servidor.
  - **creaTabYServicio():** crea las pestañas ContactsActivity, MapsActivity y ChatActivity, e inicia el servicio ServiceContacta.

Esta actividad hereda de `android.app.TabActivity`, lo que le permite albergar tantas actividades como se quiera en pestañas. Para conseguir este propósito, se crea el archivo main.xml, que será la vista de ContactaTab, en la que el elemento principal será un `android.widget.TabHost`, que va a ser el contenedor de las actividades. Dentro del TabHost, habrá un objeto TabWidget, en la que se definirá el diseño de las pestañas, y un FrameLayout, que será el layout donde aparezca la interfaz gráfica de las actividades. La estructura de la vista de ContactaTab es la siguiente:

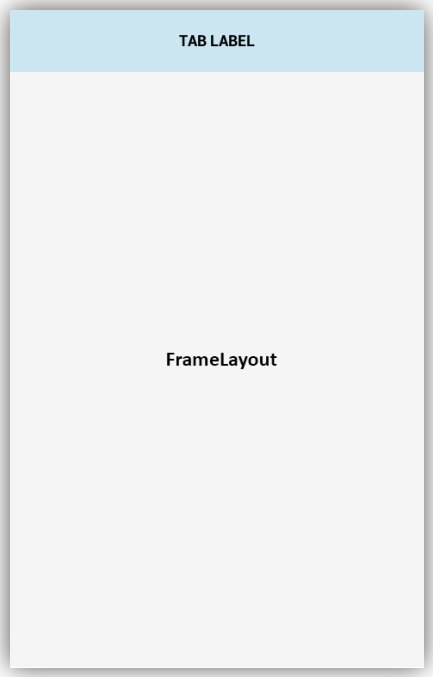


Figura 4.9: Vista de ContactaTab

Para diseñar las pestañas, se crea un fichero XML donde se define qué información aparecerá en cada una. A continuación se puede ver el código del fichero para que cada vista muestre un icono y el nombre de cada actividad:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/tabsLayout"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:background="@drawable/tab_bg_selector"
   android:gravity="center"
   android:padding="7dip"
   android:orientation="vertical">

      <ImageView
         android:id="@+id/tabsImage"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"/>

      <TextView
         android:id="@+id/tabsText"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Title"
```

Finalmente, para asignar cada actividad al TabWidget y éste al TabHost:

```
TabHost tabHost = (TabHost) findViewById(android.R.id.tabhost);
tabHost.setup();
View tabview = createTabView(tabHost.getContext(), NOMBRE_ACTIVIDAD, ICONO_ACTIVIDAD);
TabSpec setContent = tabHost.newTabSpec(tag).setIndicator(tabview).setContent(intent);
tabHost.addTab(setContent);

private View createTabView(final Context context, final String text, int drawable) {
   View view = LayoutInflater.from(context).inflate(R.layout.tabs_bg, null);
   TextView tv = (TextView) view.findViewById(R.id.tabsText);
   ImageView iv = (ImageView) view.findViewById(R.id.tabsImage);
   tv.setText(text);
   Drawable imagen = this.getResources().getDrawable(drawable);
   iv.setImageDrawable(imagen);
```

Cuando un usuario nuevo abre la aplicación por primera vez, se abre un diálogo de alerta para que introduzca su número de teléfono y poder registrarse. Esto se consigue fácilmente guardando un valor booleano en las preferencias compartidas, que por defecto esté a “false” y cuando el usuario se registre se cambie a “true”.

### 4.3.4 ContactsActivity

- **Paquete:** com.edu.contacta.ContactsActivity
- **Operaciones:**
  - **obtenerNombresYFotos():** Obtiene de la base de datos local los nombres y fotos los contactos a mostrar.

Esta actividad muestra en una lista con dos secciones. En la primera, los contactos del usuario que tengan instalada la aplicación, obteniéndose a partir de la base de datos local (cuyo procedimiento ya se ha explicado) y, en la segunda, el resto de la agenda, que se consigue haciendo una búsqueda en la base de datos de contactos del teléfono, como se demuestra a continuación:

```
public Cursor obtenerNombresYFotos(){
   String[] projection = new String[] { //Parámetros a buscar en la BBDD
                         Phones.NUMBER,
                         Phones.PERSON_ID,
                         People.NAME,
                         People._ID
                         };
   String selection = Phones.NAME + "!='null'"; //Selección de nombres no nulos
   String sortOrder = Phones.NAME + " COLLATE LOCALIZED ASC";
   Cursor cursor = managedQuery(Phones.CONTENT_URI, projection, selection, null,
sortOrder); //Búsqueda en la BBDD de contactos con la configuración deseada
   return cursor;
}
```

Para dibujar la vista, lo principal es añadir un Listview, un elemento encargado de mostrar una lista de elementos con scroll vertical a partir de un Adapter, un objeto cuya finalidad es proporcionar acceso a los datos y crear una vista para cada elemento de la lista de datos; éste lee los contactos de los dos cursores de las dos bases de datos, y los muestra.

La vista de ContactsActivity es la siguiente:

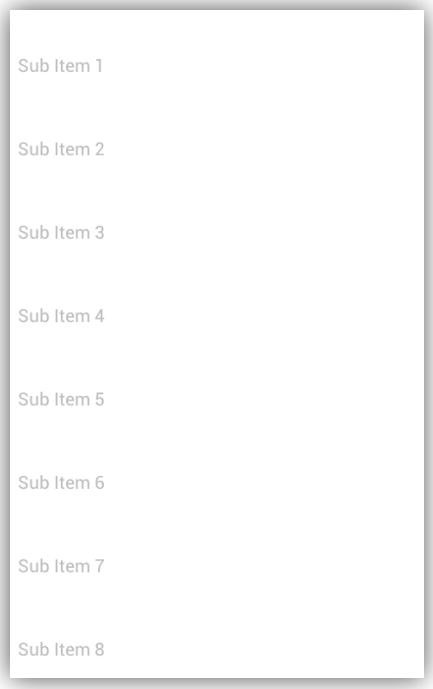


Figura 4.10: Vista de ContactsActivity

Los elementos de la lista anterior, que proporciona el Adapter, se dibujan de la siguiente manera:


Figura 4.11: Vista de elementos de la lista de contactos

En ella, aparecerá la foto, el nombre y número del contacto, y según sea un contacto con la aplicación instalada, se mostrará un CheckBox, en el que se definirá la alerta de proximidad.

Sobre estos elementos, se crea un onLongClickListener, que se encarga de escuchar los click largos. Una vez producido uno, se abre un menú contextual en el que se da la opción al usuario de llamar, enviar un SMS o un mensaje de chat. Para que esto ocurra, se registra el ListView para el menú contextual, se crea un fichero XML definiendo las entradas del menú y se sobrescriben los métodos onCreateContextMenu (para especificar qué menú ha de abrirse) y onContextItemSelected (para ejecutar una acción concreta cuando se seleccione una entrada).

### 4.3.5 Contactos

- **Paquete:** com.edu.contacta.Contactos
- **Atributos:**
  - **nombre:** Nombre del contacto.
  - **numero:** Teléfono móvil del contacto.
  - **foto:** Foto del contacto.
  - **estado:** Estado del contacto.
  - **latitud:** Latitud a la que se encuentra el contacto.
  - **longitud:** Longitud a la que se encuentra el contacto.
  - **altitud:** Altitud a la que se encuentra el contacto.
  - **ubicar:** Determina si se muestra la posición del contacto.
- **Operaciones:**
  - **get():** Getter de cada parámetro.
  - **set():** Setter de cada parámetro.

### 4.3.6 MapsActivity

- **Paquete:** com.edu.contacta.MapsActivity
- **Operaciones:**
  - **pintaEnMapa():** Dibuja en un mapa dónde se encuentran los contactos que tengan habilitada su ubicación.
  - **realidadAumentada()**: Activa el modo de realidad aumentada, abriendo la vista de la cámara y dibujando en ella las fotos de los contactos con ubicación mostrable.
- **Dependencias**:
  - **Mixare**: Librería de código abierto que gestiona la Realidad Aumentada.
  - **Cámara**: módulo que gestiona la cámara del dispositivo.
  - **GPS:** módulo que gestiona el GPS del dispositivo, permitiendo conocer la ubicación.

Esta actividad hace uso de los mapas que proporciona Android, que utilizan el motor de Google. Para hacerlos funcionar, es necesario obtener el hash de la clave de firmado que se utiliza para generar el apk de la aplicación. Una vez obtenido, se registra en la web de Google Maps para Android, y ésta genera una clave de API, la cual hay que introducir en la aplicación a la hora de configurar los mapas.

Para poder mostrar las imágenes de los contactos indicando su posición, es necesario crear una clase que gestione dicha funcionalidad. Lo mismo ocurre si se quiere añadir el gesto de hacer zoom en el mapa mediante doble click. En el siguiente diagrama de clases se podrá observar con detalle las clases que usa MapsActivity:

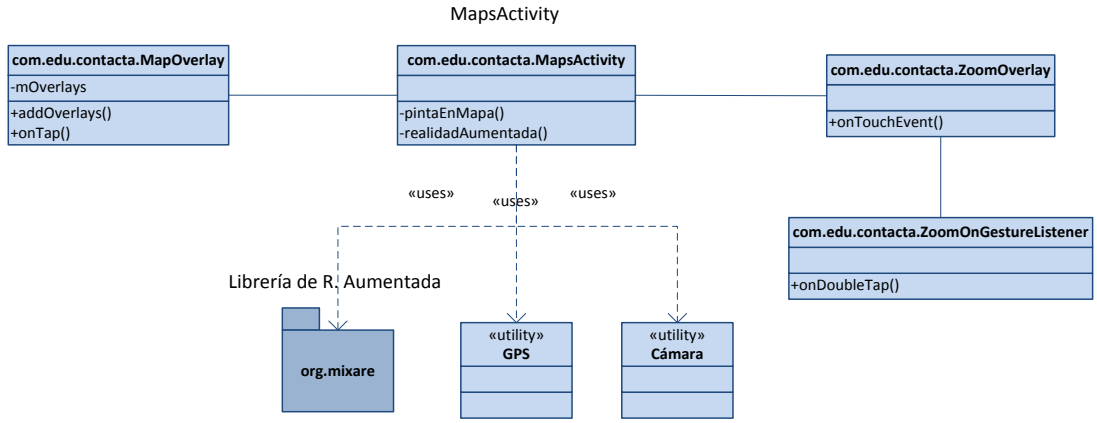


Figura 4.12: Diagrama de clases del mapa

- **MapOverlay:** se encarga de almacenar los elementos que se superpondrán en el mapa (overlays) para posteriormente dibujarlos.
- **ZoomOverlay:** crea un detector de gestos sobre el mapa.
- **ZoomOnGestureListener:** escucha el gesto de doble click hecho sobre el mapa y aumenta el zoom.

La vista asociada a MapsActivity simplemente tiene una instancia de la clase `com.google.android.maps.MapView`, que es la encargada de representar los mapas:

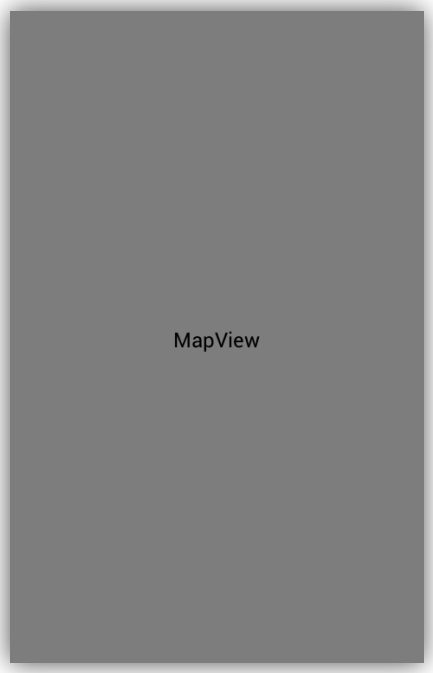


Figura 4.13: Vista de MapsActivity

Desde esta actividad, se podrá llamar a la librería Mixare de Realidad Aumentada para que represente sobre la vista de la cámara del teléfono dónde se encuentran los contactos que tengan activada la localización. También añade en la esquina superior derecha un radar en el que el centro es el propio usuario, y se ve representada la posición de los contactos y permite controlar el radio de alcance:

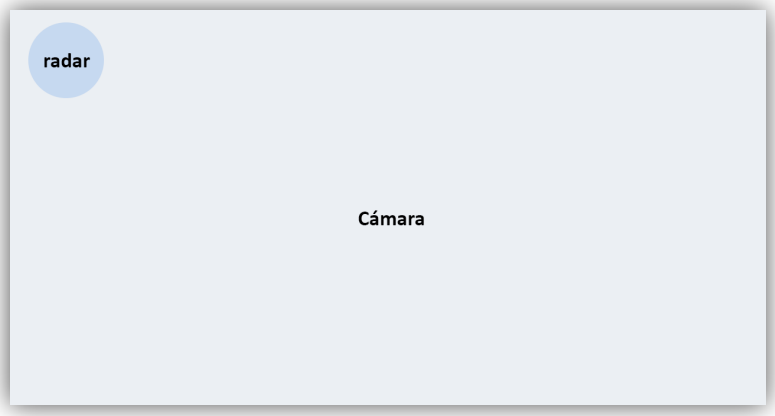


Figura 4.14: Vista de RealidadAumentada con la librería Mixare

### 4.3.7 ChatActivity

- **Paquete:** com.edu.contacta.ChatActivity
- **Operaciones:**
  - **obtenerConversaciones():** Hace una query a la base de datos local y obtiene las conversaciones de chat que haya mantenido el usuario.

ChatActivity es una actividad que muestra en una lista las conversaciones que se tengan con los contactos. Como se ha explicado anteriormente, esto requiere que se lean los datos de la base de datos local, se carguen en un Adapter y se muestren en un ListView. La vista principal coincide con la de ContactsActivity, mientras que sus elementos son distintos:


Figura 4.15: Vista de los elementos de ChatActivity

### 4.3.8 ConversationActivity

- **Paquete:** com.edu.contacta.Preferences
- **Atributos:**
  - **contacto:** Contacto con el que se está manteniendo la conversación de chat.
- **Operaciones:**
  - **sendMessage():** Una vez escrito el mensaje, este método se encarga de enviarlo al servidor, con destino el contacto de la conversación.

Una vez se hace click en un elemento de ChatActivity, se abre ConversationActivity, una actividad cuya vista se abre en un diálogo (se define en el fichero AndroidManifest.xml), quedando en el fondo la anterior actividad. Se puede ver, de nuevo en un ListView, una lista con los mensajes de chat enviados y recibidos con un contacto determinado. También, hay un cuadro de texto donde se puede enviar un nuevo mensaje.

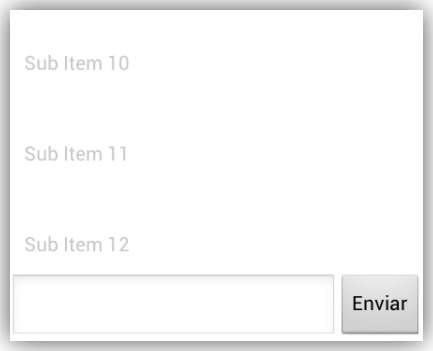


Figura 4.16: Vista de diálogo de ConversationActivity


Figura 4.17: Vista de los elementos de ConversationActivity

# Capítulo 5: Caso de estudio

*Este capítulo tiene como objetivo mostrar cómo funciona la aplicación desarrollada en un caso real, permitiendo comprender el funcionamiento de la arquitectura tanto del cliente como del servidor. Para ello, se comenzará con una descripción del escenario en el que sucede el caso de estudio elaborado y posteriormente se explicarán las acciones que se pueden llevar a cabo.*

## 5.1 Escenario

El escenario de la figura 5.1 muestra a una serie de grupos de usuarios que tienen la aplicación instalada en su teléfono y algunos de ellos tienen amistad, por lo que se tienen respectivamente en sus agendas de contactos. Para explicar el caso de estudio, se va a suponer que uno de estos usuarios se instala la aplicación por primera vez.

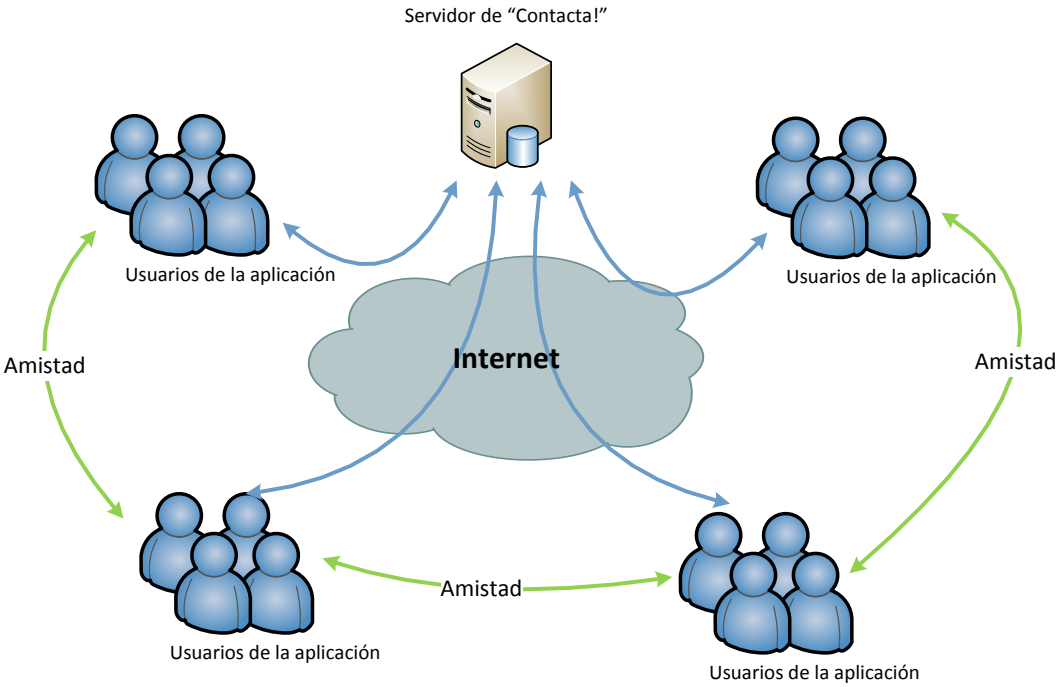


Figura 5.1: Escenario del caso de estudio

Para que todo funcione correctamente, el servidor de la aplicación tiene que estar disponible para todos, por lo que se ha decidido desplegar el Servlet en la plataforma Heroku, ya que permite realizar pruebas con mucha facilidad y sin ningún tipo de coste.

## 5.2 Acciones posibles

Ahora se explicará paso a paso todas las acciones posibles que puede realizar el usuario del escenario planteado.

### 5.2.1 Registro del número de teléfono

Nada más instalar la aplicación, si se abre aparecerá un diálogo de bienvenida indicando que se introduzca el número de teléfono para poder hacer el registro en el servidor.


Figura 5.2: Introducción de número de teléfono para registro

Una vez se introduzca un número válido, éste se registra en la base de datos del servidor, quedando vinculado a la aplicación; posteriormente se inicia el servicio que procede a adquirir la ubicación del usuario y se envía al servidor; finalmente, se descargan los datos de los usuarios del servidor y se comprueban con la agenda de contactos para ver con cuáles se tiene amistad.

### 5.2.2 Visualizar contactos en lista

La primera pantalla que se ve es la pestaña de contactos. En ella se muestra una lista con dos secciones: en la primera aparecen los contactos del usuario que tengan la aplicación instalada, en la que se muestra su foto, nombre y estado, y en la segunda el resto de contactos de la agenda, mostrándose la foto, nombre y número de teléfono.

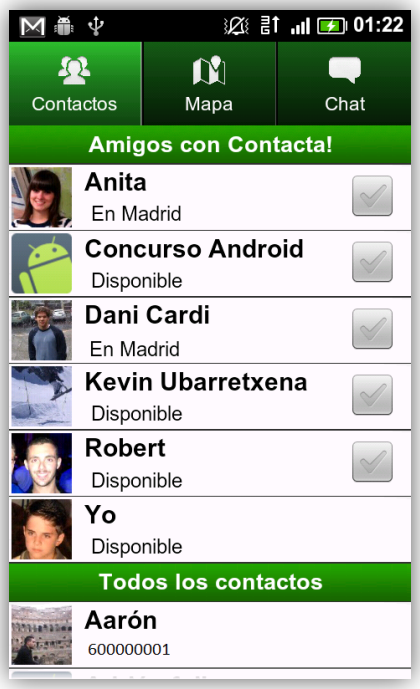


Figura 5.3: Lista de contactos

Si se pulsa el botón menú del teléfono en esta pantalla, se podrá actualizar, ir a preferencias o salir de la aplicación. Esto se podrá hacer desde cualquier pestaña.


Figura 5.4: Menú de opciones en lista de contactos

Dejando pulsado un contacto, se abre un menú contextual en el que se podrá seleccionar si llamar a un contacto, enviar un SMS o un mensaje de chat (si dicho contacto dispone de la aplicación).

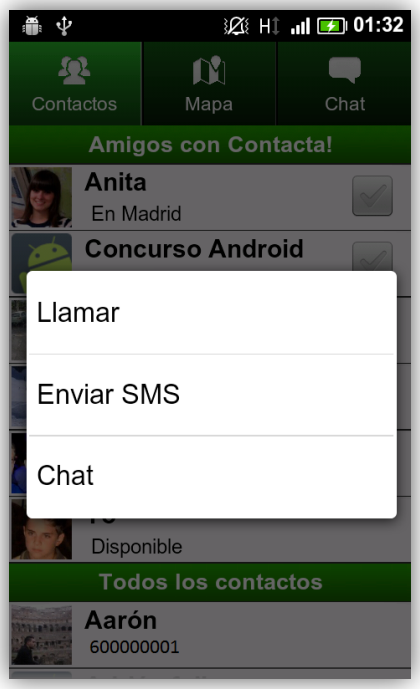


Figura 5.5: Menú contextual de un contacto

### 5.2.3 Visualizar contactos en el mapa

Si se accede a la segunda pestaña, se muestra un mapa de Google Maps en el que se dibuja la foto de los contactos que tengan la aplicación en su ubicación más reciente. También se muestra la foto y ubicación del propio usuario (las fotos se obtienen de la agenda de contactos).

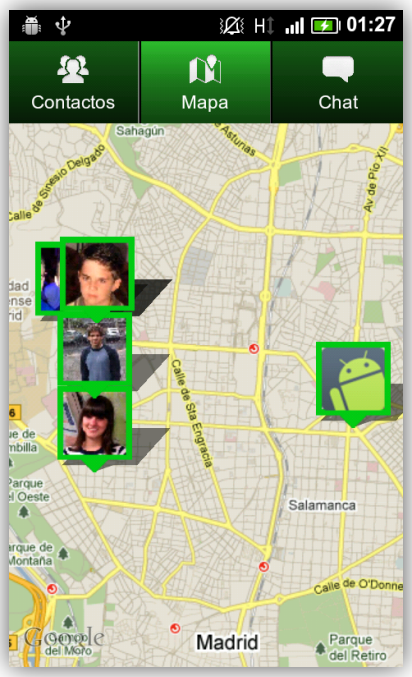


Figura 5.6: Vista de los contactos en el mapa

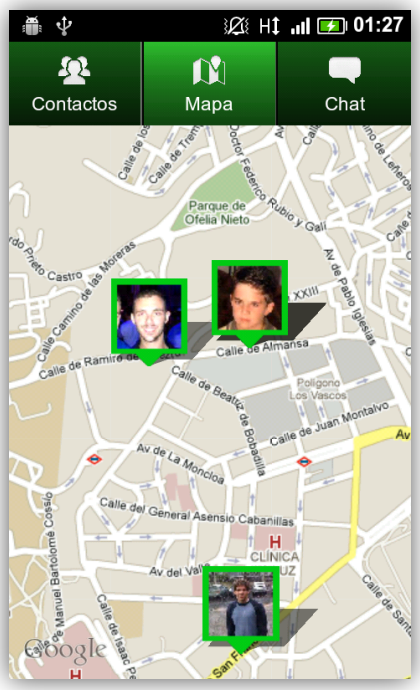


Figura 5.7: Zoom en vista de mapa

### 5.2.4 Visualizar contactos en Realidad Aumentada

En la vista del mapa, pulsando el botón menú del terminal, se muestra la opción de Cámara. Si se selecciona, se activa la vista de la cámara, en la que se ven dibujados las fotos de

los contactos según su posición más reciente, junto con su nombre y distancia. Según se vaya moviendo el teléfono, se irá corrigiendo la posición de las imágenes en pantalla, indicando en todo momento la dirección donde se encuentra cada contacto.

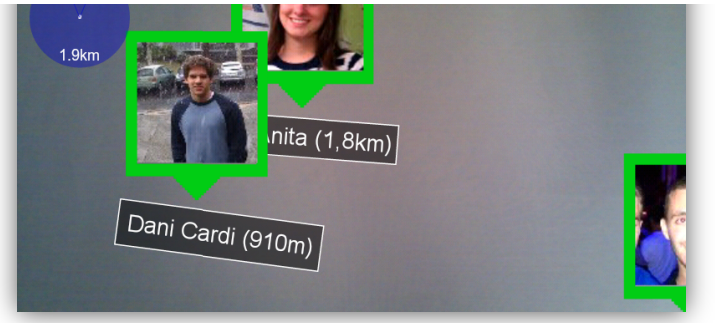


Figura 5.8: Vista de Realidad Aumentada

En la esquina superior izquierda se puede observar un radar, donde se muestra el radio de visión de la cámara, qué dirección se está viendo y dónde se encuentran los contactos. Si se pulsa el botón de menú, aparece la opción de Zoom, en la que se puede configurar el radio de visión desde 0km hasta 80km.

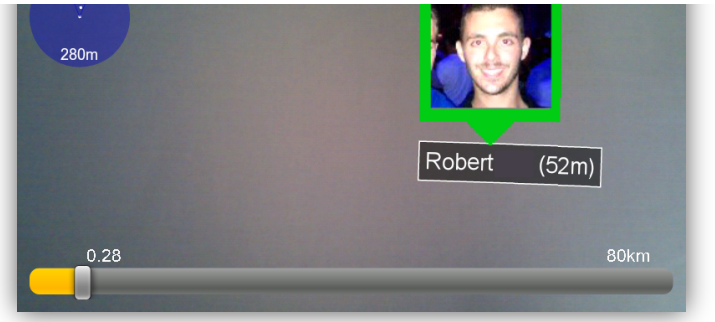


Figura 5.9: Ajuste del radio de visión

### 5.2.5 Vista de conversaciones de chat

La tercera pestaña muestra las conversaciones de chat que se hayan mantenido con los contactos que dispongan de la aplicación. De cada conversación se muestra la foto del contacto, su nombre y el último mensaje que se ha escrito.

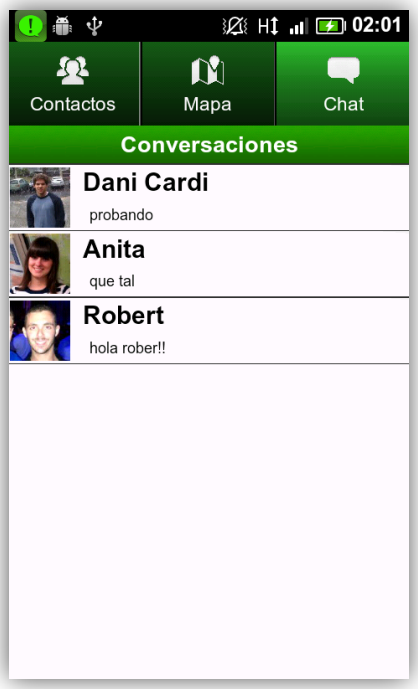


Figura 5.10: Lista de conversaciones de chat

Haciendo click en una de ellas, se abre una ventana en la que se ven los mensajes de la conversación y se da la opción de enviar uno nuevo.

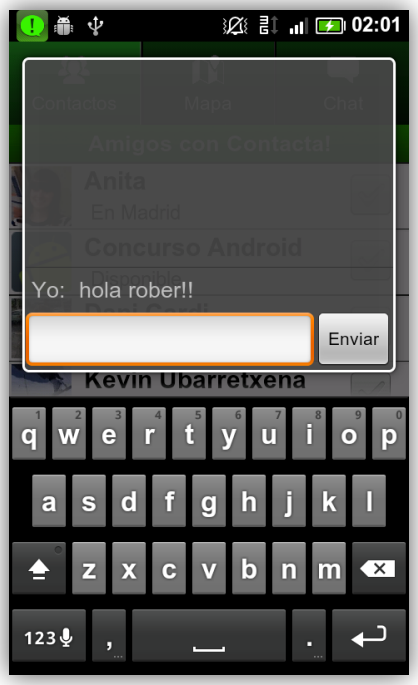


Figura 5.11: Conversación de chat con un contacto

Al recibir un mensaje de chat, la aplicación generará una notificación indicando que ha llegado un nuevo mensaje, y haciendo click se abre la conversación.


Figura 5.12: Notificación de chat recibido


Figura 5.13: Mensaje recibido en la conversación

### 5.2.6 Menú de preferencias

Para abrir la pantalla de preferencias, En cualquiera de las tres pestañas, si se abre el menú contextual a través del botón menú del teléfono y se selecciona la opción de preferencias.


Figura 5.14: Preferencias de la aplicación

Se puede controlar si la aplicación se ejecuta al encender el terminal, si se muestra la ubicación a los contactos, la frecuencia de actualización del servicio, el radio de proximidad para las alertas y el estado.

### 5.2.7 Alertas de proximidad

Para poder establecer alertas de proximidad con un contacto, es necesario activar su checkbox, que está situado en la pestaña de contactos y definir el radio de proximidad deseado. Una vez hecho esto, la aplicación irá comprobando periódicamente (según el valor de actualización indicado en las preferencias) si dicho contacto se encuentra dentro de la distancia máxima establecido. Si esto ocurre, se genera una notificación indicando quién se encuentra cerca y a qué distancia en metros.

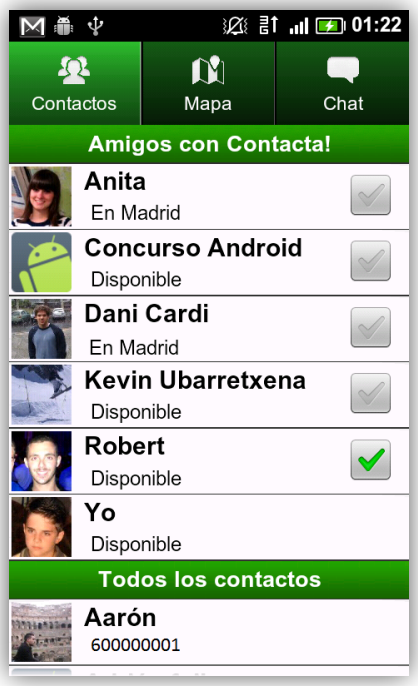


Figura 5.15: Checkbox del contacto a notificar activado


Figura 5.16: Notificación de proximidad recibida

## 5.3 Servidor: peticiones y respuestas

Vistas ya las posibles acciones a realizar desde el cliente, a continuación se va a mostrar cómo se llevan a cabo las peticiones al servidor y las respuestas que éste genera, que son las que hacen posible que funcione la aplicación.

Se van a representar dos casos: el registro de un usuario, que engloba también a la actualización de datos, obtención de la información de contactos y descarga de mensajes de chat, y el envío de un mensaje de chat. Con estos dos casos, quedan ejemplificadas todas las llamadas que conciernen a la aplicación.

### 5.3.1 Logs del registro de usuario

El intercambio de mensajes que se realiza cuando un usuario se registra es el siguiente:

1. El usuario hace la petición de registro con su número de teléfono como identificador.
2. Una vez que se ha realizado el registro, se hace un update añadiendo la información del usuario, como su localización y el estado.
3. Se pide al servidor la información de los contactos.
4. Descarga los mensajes de chat que tenga pendientes.

Cuando un usuario abre la aplicación, se realiza el mismo intercambio de mensajes a partir del punto 2. En el siguiente ejemplo, se pueden ver los logs del servidor mostrando los mensajes comentados anteriormente:

```
2013-07-11T15:43:53.360959+00:00 app[web.1]: ContactaServer: (GET
/contacta?Registro=si&Numero=686478580)
2013-07-11T15:43:53.361324+00:00 app[web.1]: Registrando...
2013-07-11T15:43:53.938636+00:00 app[web.1]: Filas cambiadas: 1
2013-07-11T15:43:53.874365+00:00 app[web.1]: ContactaServer: (GET
/contacta?Update=si&Numero=686478580&Latitud=40.4510368&Longitud=-
3.712512&Altitud=0.0&Precision=91.358&Estado=Disponible&Ubicar=true)
2013-07-11T15:43:53.874549+00:00 app[web.1]: Actualizando...
2013-07-11T15:43:53.938636+00:00 app[web.1]: Filas cambiadas: 1
2013-07-11T15:43:54.314336+00:00 app[web.1]: ContactaServer: (GET
/contacta?Coordenadas=si)
2013-07-11T15:43:54.361911+00:00 app[web.1]:
[{"Ubicar":"true","Estado":"Disponible","Latitud":"40.4510306","Numero":"600000001","Preci
sion":"36.164","Altitud":"0.0","Longitud":"-
3.7125155"},{"Ubicar":"true","Estado":"Disponible","Latitud":"40.4510771","Numero":"600000
004","Precision":"0","Altitud":"0.0","Longitud":"-
3.7125163"},{"Ubicar":"true","Estado":"Disponible","Latitud":"40.4510368","Numero":"686478
580","Precision":"91.358","Altitud":"0.0","Longitud":"-
3.712512"},{"Ubicar":"true","Estado":"Anita","Latitud":"40.4509188","Numero":"664708225","
Precision":"10","Altitud":"70","Longitud":"-3.7129586"}]
2013-07-11T15:43:54.621756+00:00 app[web.1]: ContactaServer: (GET
/contacta?RecibirChat=si&Destino=686478580&Hora=0)
```

### 5.3.2 Logs al enviar un mensaje de chat

Si un usuario envía un mensaje de chat a otro, envía la siguiente petición al servidor:

```
2013-07-11T16:05:09.405251+00:00 app[web.1]: ContactaServer: (GET
/contacta?EnviarChat=si&Destino=600000004&Mensaje=Hola,probando&Origen=686478580
)
2013-07-11T16:05:09.489950+00:00 app[web.1]: Filas cambiadas: 1
```

# Capítulo 6: Conclusiones y trabajos futuros

*En este capítulo se van a presentar las conclusiones y recomendaciones que se han obtenido una vez realizado el proyecto. Además, se incluyen también las posibles mejoras que podrían realizarse para poder perfeccionar las características tanto del cliente como del servidor y obtener un mejor y más completo sistema.*

# 6.1 Conclusiones y recomendaciones

El principal objetivo de este proyecto es el desarrollo de una aplicación móvil consistente en una red social formada por los contactos que contiene la agenda del móvil, en la que se geolocalizan los usuarios en un mapa y a través de Realidad Aumentada. Una vez terminado el proyecto, se presentó al "II Concurso de Desarrollo de Aplicaciones para Android", promovido por Telefónica y la Cátedra Telefónica Aplicaciones y Servicios Móviles en la Universidad Politécnica de Madrid, obteniendo el segundo premio.

Esto ha sido posible desarrollarlo gracias a los beneficios que ofrecen tanto Android como los Servlets, necesarios para gestionar toda la parte del servidor. Tanto para la aplicación cliente como para el servidor, se utiliza el lenguaje de programación Java, por lo que se recomienda encarecidamente utilizar dicha combinación ya que ofrece muchas sinergias.

Para desplegar el Servlet, se ha utilizado la plataforma de computación en la nube Heroku, que tiene completo soporte para ello y está documentado a la perfección. Si se utiliza el sistema más básico es completamente gratuito, por lo que es ideal para este tipo de prácticas; teniendo también la opción de aumentar la capacidad en cualquier momento. Además, se le pueden incluir plugins, y entre ellos se encuentra el de PostgreSQL, que es la base de datos que se ha utilizado para realizar el servidor. Así, se convierte en la forma más sencilla para poder desarrollar una aplicación móvil de Android y comenzar a distribuirla.

El intercambio de datos entre el cliente y servidor se produce gracias al formato JSON. Tiene muchas ventajas respecto a otros formatos de intercambio, como la ligereza, facilidad de generación y lectura, etc. Además, Android incluye unas API's para ello, con lo que es más sencillo aún su uso. Para casos en los que se envíe mucha cantidad de información, se puede utilizar la librería Gzip, que junto con una serie de algoritmos es capaz de reducir el tamaño de los datos en un 99%, siendo ideal para comunicaciones con móviles, para así ahorrar batería y consumo de megas.

Existen numerosas formas de geolocalizar a usuarios en un mapa en una aplicación de Android, pero sin duda la mejor es mediante Google Maps. Entre sus numerosas ventajas cabe nombrar que viene integrado  defecto, sus API's son muy sencillas de utilizar y están bien documentadas, carga muy rápido, su contenido es muy fiable, etc. Otras soluciones, como Open Street Maps, tienen características que le faltan a los mapas de Google pero son más complicadas de integrar y su aspecto visual es muy pobre, por lo que claramente es preferible utilizar los primeros.

La Realidad Aumentada es una de las funcionalidades principales de la aplicación. Ha sido posible implementar esta funcionalidad gracias a Mixare, una librería dedicada a tal propósito gratuita y de código libre. Existen otras soluciones, pero la mayoría requieren licencia para utilizarlas.

## 6.2 Trabajos futuros

Durante la realización del proyecto y una vez finalizado se han observado y analizado las posibles mejoras y ampliaciones para un futuro. A continuación, se detallan las más importantes:

- **Adaptar la aplicación cliente a la última versión de Android.**
  Cuando comenzó a desarrollarse el proyecto, la versión más utilizada de Android era la 2.3.3. Actualmente esto ha cambiado, y actualmente la mayoría de los dispositivos tiene la 4.0 o superior. Por tanto, sería conveniente adaptar la aplicación a los nuevos estándares para aprovechar al máximo las posibilidades que éstos ofrecen.

- **Cambiar la interfaz gráfica de la aplicación cliente y adaptarla al estándar "Holo".**
  Una de las importantes mejoras que introdujo Honeycomb (Android 3.0), disponible también en las versiones superiores, es "Holo", una nueva interfaz gráfica que trata de hacer sentir cómodo al usuario y que le resulte algo natural, intuitivo, manejable y fácil de aprender. Con esta nueva interfaz, se desarrollaron unas guías de diseño con la finalidad de que todas las aplicaciones adoptaran de la misma manera la nueva apariencia. Adoptando estos nuevos principios, la aplicación obtendría una considerable mejora visual y haría más sencillo su uso.

- **Introducir las notificaciones de "Google Cloud Messaging for Android".**
  Para reducir el gasto de batería y aumentar la velocidad de respuesta de la aplicación, se podría implementar el servicio de notificaciones que proporciona Google para Android. Así, el servicio que posee el cliente para preguntar por datos nuevos cada cierto tiempo sería sustituido por este sistema, en el que todo el trabajo lo asume el servidor, notificando a cada cliente cuando tiene nuevo contenido.

- **Actualizar a la versión v2 de la API de Google Maps.**
  Como uno de los principales servicios que ofrece la aplicación está basado en los mapas, sería recomendable actualizar a la siguiente versión de la librería que gestiona Google Maps para Android. Ésta proporciona nuevas funcionalidades, como mapas en 3D, opciones de rotación y cambio de perspectiva, posibilidad de insertar mapas propios, mejor gestión de introducción de elementos, etc.

- **Mejorar la seguridad del servidor.**
  Para que ningún usuario obtenga datos que no le correspondan, sería necesario aplicar una capa de seguridad a las comunicaciones con el servidor, con la que un usuario sólo pudiera obtener sus propios datos. Esto podría lograrse con un sistema de autenticación basado en Tokens, generados por ejemplo a partir del número de teléfono del usuario y una característica propia de su terminal, como el IMEI, la MAC de la tarjeta inalámbrica etc.

- **Añadir verificación de números de teléfono.**
  Actualmente, en el registro del número de teléfono del usuario con la aplicación y el servidor no se verifica que éste sea suyo propio, por tanto cualquier usuario podría registrarse con un número que no le corresponda. Para paliar este defecto, habría

que añadir un sistema de verificación, ya sea enviando un SMS a dicho número con un código (más caro) o enviando un email de confirmación (más barato).

# Bibliografía


[1] *Android, www.android.com.*

[2] *Tienda de aplicaciones de Android, Google Play, http://play.google.com/store/apps.*

[3] Stephanie Bodoff, Eric Armstrong, Jennifer Ball, Debbie Bode Carson, Ian Evans, Dale Green, Kim Haase, Eric Jendrock. *The J2EE™ Tutorial Second Edition,* 2004.

[4] Budi Kurniawan. *Servlet and JSP: A Tutorial,* 2012.

[5] Toby J. Teorey; Sam S. Lightstone; Tom Nadeau; H.V. *Jagadish. Database Modeling and Design, 5th Edition,* 2011.

[6] S G Ganesh, Tushar Sharma. *Oracle Certified Professional Java SE 7 Programmer Exams 1Z0-804 and 1Z0-805: A Comprehensive OCPJP 7 Certification Guide*, 2013.

[7] Mike Keith, Merrick Schincariol. *Pro JPA 2: Mastering the Java™ Persistence API,* 2009.

[8] David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, Sailu Reddy. *HTTP: The Definitive Guide,* 2002.

[9] *Introducing JSON, www.json.org*.

[10] Marko Gargenta. *Learning Android*, 2011.

[11] Michael Miller. *Using Google Maps and Google Earth*, 2011.

[12] Greg Kipper, Joseph Rampolla. *Augmented Reality*, 2012.

[13] Raghav Sood. *Pro Android Augmented Reality*, 2012.

[14] *Mixare, Free Open Source Augmented Reality Engine, www.mixare.org.*