Departamento de Ing. Telemática

PROYECTO FIN DE CARRERA

Desarrollo de una aplicación móvil de compra de entradas adaptativa según contexto para Android

Ingeniería de Telecomunicación

Autor: Javier Vaíllo Martín

7Tutor: Florina Almenares

Leganés, 22 octubre de 2013

Título:
Autor:
Director:

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Al fin se acaba esta andadura por el mundo universitario. Han pasado muchos años y en muchas ocasiones ha sido muy difícil ver la luz al final del túnel, pero gracias al apoyo de muchas personas he conseguido llegar hasta dónde estoy ahora.

Lo primero de todo es agradecer a mi familia todo el apoyo y esfuerzo que han puesto en mí y cómo me han intentado dar el empujón que necesitaba para seguir adelante. A mi padre por la paciencia que ha tenido que desarrollar conmigo, a mi madre por darme todo sin esperar nada, a mi hermana por darme aliento y ejemplo. Sé que se os ha hecho tan difícil como a mí, muchísimas gracias por todo.

A Sheila por estar siempre ahí y aguantarme en mis peores momentos. Sé que has sufrido mucho por mi ausencia, pero a partir de hoy tenemos todo el mundo para nosotros.

A todos los amigos que he conocido en la universidad y que han trascendido en mi vida. A Miki por todas las horas incontables que hemos pasado juntos entre apuntes, prácticas y escapadas. A Natalia por intentar inculcarnos el sentido común cuando lo hemos necesitado y dejarse llevar (o tirar de nosotros) cuando tocaba. A Dani, que aunque llegó más tarde, en seguida se vio que era carne de Griffindor. A Rosa que siempre ha estado ahí con una sonrisa y para lo que hiciese falta. A Kiko, Santi, Carlos, Fer, Manu, Javo, Raúl, Luís, Rober, Bea y otros como Mario o Víctor que tuvieron que caer a la técnica, muchas gracias por todos esos momentos que hemos pasado juntos, esos viajes a Gandía o a Galicia, esos miércoles de realimentación y tantos instantes que hemos compartido y que aún nos quedan por compartir.

Gracias a todos mis amigos de Alcorcón por no desesperarse al escuchar eso de “no salgo, que tengo que estudiar” y seguir año tras año, y ya van muchos, recorriendo el camino juntos.

Para terminar quería agradecerle a Florina el esfuerzo tan grande que ha hecho para que pueda estar hoy presentando el proyecto. Perdona por el estrés final y muchísimas gracias.

Resumen

En la actualidad el uso de teléfonos móviles inteligentes o *Smartphones* ha revolucionado la forma en la que se comunican las personas entre sí, la manera en la que interactúan con su entorno y el modo de entender las comunicaciones en general.

Este tipo de terminal se caracteriza por una capacidad de procesamiento similar a la de un ordenador portátil, un acceso de alta velocidad a Internet y la utilización de sensores integrados como sistemas de geolocalización, acelerómetros, sistemas de reconocimiento por voz, etc. Gracias a estas características surge la posibilidad de instalar aplicaciones adicionales en el dispositivo, que proporcionan un gran abanico de oportunidades entre las que un usuario puede encontrar diversos tipos de funcionalidad.

Entre los distintos tipos de aplicaciones se encuentran aquellas que están basadas en la posición actual del usuario, denominadas LSB por sus siglas en inglés (*Location Based Services*). La motivación principal de este tipo de aplicaciones es proporcionar al usuario información acerca de su entorno, como por ejemplo los restaurantes cercanos o la forma de llegar a una determinada dirección.

Dentro de este tipo de servicios se enmarca el presente proyecto, en el que se va desarrollar una aplicación móvil para el sistema operativo Android, que permita la compra de entradas personalizada (a través de distintos sensores embebidos en el terminal) para distintos eventos de ocio, tales como obras de teatro, sesiones de cine, multiaventuras, etc. En concreto, el prototipo desarrollado se ha realizado utilizando cines y teatros.

Existen dos principales diferencias entre esta aplicación y las similares ya existentes. La primera es la integración de la información obtenida de los eventos en un mapa, con la opción de cálculo de rutas y presentación de una interfaz personalizada de acuerdo a información de contexto relacionada con el usuario. La segunda es el planteamiento adaptativo al usuario que, además de poder personalizar la interfaz y utilizar su posición exacta como un motor inicial de búsqueda de los eventos, crea un perfil propio del usuario. En este perfil se almacenarán sus preferencias sobre el aspecto y funcionalidad de la aplicación y se monitorizarán sus elecciones sobre los eventos para dar prioridad a aquellos que se adapten a sus preferencias y personalizar así la cartelera de los mismos.

Abstract

Nowadays the use of smart phones has revolutionized the way in which people communicate with each other. As well, it has changed the way we interact with our environment and how to understand communications in general.

This type of terminal is characterized by a processing capability similar to a laptop computer, a fast access to the Internet and use of integrated sensors and location systems, accelerometers, voice recognition systems, etc.. Thanks to these features comes the ability to install additional applications on the device, which provide a wide range of opportunities from which the user can find various types of functionality.

Among the different types of applications, there are some based on the user's current position. They are known as “LSB” (Location Based Services). The main motivation for this type of application is to provide the user with information about their environment, such as nearby restaurants or how to get to a certain direction.

This project belongs to this kind of services. It consists of the development of a mobile application for Android operating system, which allows to buy personalized ticket (through various sensors embedded in the terminal) for several entertainment events such as plays, film sessions, adventure sports, etc.. In particular, the prototype developed has been performed using cinemas and theaters.

There are two main differences between this application and the similar existing. Firstly, the integration of information from events on a map, with the option of route calculation and presentation of the customer interface according to information related to the user context. The second approach is the adaptation to the user, as well as to customize the interface and use their exact position as an initial engine to search events, it creates an unique user profile. In this profile, preferences will be stored on the look and functionality of the application and monitored their choices on events to give priority to those that suit the preferences and also customize the way the events are shown.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Motivación del proyecto

En la actualidad el uso de teléfonos móviles inteligentes o smartphones ha revolucionado la forma en la que se comunican las personas entre sí, la manera en la que interactúan con su entorno y el modo de entender las comunicaciones en general.

Un Smartphone se puede definir como un teléfono móvil con una capacidad de procesamiento, conectividad y almacenamiento de datos superior a la de un teléfono móvil convencional. Este tipo de terminal actualmente tiene una capacidad de procesamiento similar a la de un ordenador portátil y su uso está cada vez más extendido. Actualmente la penetración en el mercado de la telefonía móvil de los terminales de tipo Smartphone en España supone un 66% del total de la cuota [1], de los cuales el 92% tienen una versión del sistema operativo Android [2]. Además, se consolida el uso de otros dispositivos portátiles como las tabletas, cuya distribución se espera que supere a la de los portátiles durante el presente año [3]. Todos estos indicadores reflejan la forma en la que estos dispositivos se introducen en la vida cotidiana y se convierten en una herramienta a la que cada día se le da más uso.

La característica más importante de un Smartphone es una buena conectividad a internet. La implantación y expansión de las redes UMTS de telefonía 3G propulsó la creación de estos terminales, dadas las infinitas posibilidades que ofrecía al usuario un acceso a internet fluido en todo momento y en cualquier lugar. Posteriormente, ante el auge de la utilización de este tipo de dispositivos, las empresas de telefonía apostaron por el desarrollo de las redes móviles para ofrecer a los usuarios las mejores prestaciones con la mayor cobertura posible. De esta manera se empiezan a instalar antenas de HSPA, HSPA+ y recientemente de 4G que mejoran la velocidad de acceso de los usuarios a internet.

Otra característica clave en la evolución de los dispositivos móviles ha sido los nuevos sistemas operativos que permiten la instalación de aplicaciones adicionales en el dispositivo. Éstas pueden ser desarrolladas por el fabricante del terminal, por el operador

o por cualquier empresa de software, ofreciendo una infinidad de posibilidades para explotar las cualidades de los dispositivos y del acceso a la información.

Una de las cualidades más interesantes de los dispositivos Smartphone es la inclusión de sistemas de geoposición como el GPS que permiten obtener la ubicación del usuario en cualquier momento. Sobre esta funcionalidad se crea una modalidad de aplicación en base a la posición del usuario. Este tipo de aplicaciones se denominan servicios basados en localización o LSB, por sus siglas en inglés (Location Based Services). Estos servicios utilizan como parámetro la posición del usuario para ofrecerle información de su entorno, como por ejemplo los restaurantes cercanos o la forma de llegar a una determinada dirección.

Entre otras funcionalidades, este tipo de servicios sirven para proporcionar al usuario información práctica sobre diferentes alternativas para su tiempo libre. De esta manera se pueden obtener datos en tiempo real sobre actividades lúdicas y culturales ubicadas en el entorno próximo desde el que se encuentra el terminal. Así es posible valorar en cualquier momento las diferentes opciones de ocio que se presentan sin necesidad de una planificación previa.

Dentro de este tipo de servicios se enmarca el presente proyecto, en el que se va a desarrollar una aplicación móvil para el sistema operativo Android, que permita la compra de entradas personalizada (a través de distintos sensores embebidos en el terminal) para distintos eventos de ocio, tales como obras de teatro, sesiones de cine, o multiaventuras. En concreto, el prototipo desarrollado se ha realizado utilizando cines y teatros.

Aunque existen algunas aplicaciones para compra de entradas para eventos para terminales móviles, estas aplicaciones generalistas muestran sus resultados en base a una ciudad o provincia y ofrecen la información por medio de listas. También es posible que muestren recintos y eventos cercanos de acuerdo con la localización considerando un radio amplio, pero no ofrecen integradas las opciones de verlo en un mapa, con la opción de cálculo de ruta y además de presentar una interfaz personalizada de acuerdo a otra información de contexto relacionada con el usuario. Por tanto, la aplicación que se desarrolla en el presente proyecto aumenta la importancia del usuario utilizando, además de su posición exacta como un motor inicial de búsqueda de los eventos e inyectando toda la información obtenida en un mapa para mostrarla en función de su ubicación, la información almacenada en el perfil propio de usuario. En este perfil se almacenarán sus preferencias sobre el aspecto y funcionalidad de la aplicación, además de la información obtenida por ésta al monitorizar el tipo de eventos que selecciona el usuario para dar prioridad a los que se adapten más a sus gustos con el fin de personalizar la cartelera de eventos.

1.2 Objetivos

El objetivo general de este proyecto es desarrollar una aplicación LBS (*Location Based Service*) Android que obtenga la información personalizada en tiempo real de la web sobre los eventos que estén próximos a la posición del usuario y permita, además, acceder a la compra de las entradas.

En esta primera versión de la aplicación se va a utilizar como fuente de información de eventos el servicio web de Kulturklik [4] para obras de teatro y la página web de Google dedicada a las carteleras para sesiones de cine [5].

Los objetivos específicos que ha de cumplir la aplicación son:

- Localizar los cines cercanos a la posición de usuario en cualquier lugar de España, obtener la cartelera con las películas que se proyectan ese mismo día y los horarios de las sesiones, y poder acceder a la compra de la entrada.
- Detectar las obras de teatro que se representan ese mismo día en el País Vasco y poder acceder a la compra de la entrada.
- Mostrar los resultados en un mapa mediante marcadores ubicados en la posición del evento respecto a la del usuario.
- Posibilidad de interactuar con los marcadores del mapa para acceder a la información de los eventos a los que representan.
- Mostrar el trayecto desde el punto en el que se encuentra el usuario hasta el lugar del evento. Pintarlo sobre el mapa y ofrecerle las indicaciones escritas de cómo llegar hasta él. El cálculo del trayecto se puede realizar para tres medios de transporte distintos: a pie, en coche o en transporte público.
- Ofrecer la posibilidad al usuario de visualizar la cartelera de los cines cercanos a su posición por películas y por rango de horarios, teniendo en cuenta la hora local.
- Detectar, al obtener la información, si un usuario está en las inmediaciones de la ubicación en donde se produce uno de los eventos contemplados por la aplicación, y mostrar directamente la información de dicho evento.
- Permitir guardar las preferencias del usuario en cuanto a las opciones de visualización de mapas, tipos de búsqueda, medio de transporte para el cálculo de rutas, etc.
- Monitorizar el tipo de eventos que interesan al usuario y almacenar esta información para ordenar los resultados conforme a sus preferencias.

1.3 Fases de desarrollo

En este apartado se muestran brevemente las diferentes fases que se han realizado para el desarrollo de este proyecto. A continuación se enumeran cronológicamente.

Fase 1: Definición de requisitos

La primera fase del proyecto consistió en la definición de éste, en la que se especificaron las funcionalidades básicas que debía tener y el tipo principal de interfaz de usuario con el que se mostraría la información.

Fase 2: Búsqueda de fuentes de información

Una vez definida la idea principal, se hizo una selección de posibles eventos para incluir en la aplicación, tales como sesiones de cine, obras de teatro, conciertos, eventos deportivos, etc. Se buscaba el requisito de poder obtener información en tiempo real de la red, ya fuera mediante peticiones a servicios web o a través de información extraída de una página de Internet.

Fase 3: Despliegue del entorno de desarrollo

En esta fase se configuró el entorno de desarrollo de Eclipse para el desarrollo de aplicaciones Android en general y se añadieron los paquetes y librerías necesarios para permitir el uso de los mapas.

Fase 4: Diseño y desarrollo modular de las principales actividades

Dado que los conocimientos que se poseía sobre la programación Android eran muy básicos, se decidió crear las actividades principales por separado. De esta manera los primeros pasos consistieron en crear el esquema básico de las diferentes actividades que se iban a utilizar en la aplicación, en las que se diseñaba un interfaz y se controlaban los eventos que se producían sobre ellas.

Fase 5: Diseño y desarrollo modular de la obtención de información

Al igual que en la fase anterior, se decide implementar por separado las tareas que se conectan con la red, extraen la información requerida y las guardan en los objetos que se

han diseñado para ello y así poder utilizar esta información en el resto de la aplicación. En esta etapa se centró el esfuerzo en obtener la funcionalidad para los eventos de tipo cine.

Fase 6: Migración a Google Maps Android API v2

El presente proyecto se inició utilizando la primera versión del API de Google Maps para Android para la creación y gestión de los mapas de la aplicación. Durante el desarrollo de la aplicación, esta versión se marca como obsoleta por Google y lanza una segunda versión. Ante esta situación se decide migrar las actividades que utilizaban el mapa a la nueva versión de la API.

Fase 7: Integración de módulos

Una vez migrados los mapas de la aplicación a la nueva versión del API, se procede a la integración de los módulos que manejan las actividades y los que obtienen la información de la red. Se hacen las primeras pruebas de integración y se obtienen los primeros resultados.

Fase 8: Generalización de eventos y adición de eventos de tipo teatros

Llegados a esta fase ya se dispone de una aplicación básica que muestra las carteleras de los cines cercanos. Se generaliza el objeto que almacenaba la información de un cine creando una unidad de información básica genérica para eventos y se añade el de tipo teatro. También se desarrolla la tarea que extrae la información sobre los teatros de la red.

Fase 9: Adaptación de la aplicación al usuario

En esta fase se incluye la funcionalidad que permite al usuario guardar su configuración de la forma en la que se muestra la información, y que monitoriza sus acciones para poder ordenar los resultados en función de sus preferencias.

Fase 10: Integración y depuración

En esta fase se realizan los cambios necesarios en la interfaz y en la interoperabilidad entre los diferentes módulos, para integrar las diferentes partes de las que consta la aplicación. Después se hacen las pruebas de integración y funcionalidad para depurar errores.

Fase 11: Pruebas de usuario

En esta fase se instala la aplicación en móviles de usuarios para comprobar la experiencia de usuario: facilidad en la utilización, claridad en la exposición de los datos, sencillez en la navegación, etc.

Fase 12: Redacción de la memoria

Por último se procede a documentar el proyecto en sí mismos con la redacción de la presente memoria.

1.4 Medios empleados

En este apartado se van a indicar los medios necesarios para la realización de este proyecto:

Hardware:

- PC Intel Core 2 Quad
- Teléfono móvil HTC Legend
- Cable USB 2.0 a Micro USB

Software:

- Windows XP (sistema operativo PC)
- Android 2.2 Froyo (sistema operativo móvil)
- Eclipse 4.2 Juno (entorno de desarrollo)
- Java Development Kit 1.7.0_11 (paquete de compilación Java)
- ArgoUML 0.34 (software para la creación de diagrama de clases)
- Microsoft Word 2010 (Editor de texto memoria)
- Adobe Photoshop CS6 (Editor gráfico fondos de pantalla aplicación)
- Adobe Reader (Programa de visualización de PDF's)
- Google Chrome (Navegador web)

Otros

- Conexión a Internet

1.5 Terminología

En este apartado se detalla un glosario en el que se explican brevemente algunos términos que se han usado en el presente documento con el fin de que el lector pueda comprender el total del contenido.

1.5.1 Glosario de términos

- ***Smartphone***: Teléfono móvil inteligente que tiene mejores prestaciones que un teléfono normal y permite el acceso a Internet a gran velocidad.

- **Android**: Sistema operativo de código abierto basado en Linux especialmente diseñado para su uso en dispositivos móviles y tabletas.

- **Tableta**: Dispositivo táctil de prestaciones similares a un ordenador portátil

- **Geoposición**: Coordenadas latitud y longitud de un punto.

- **Geolocalización**: Coordenadas latitud y longitud de un punto.

- **Aplicación**: Programa informático que realiza una acción determinada

- **Servicio web**: Tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones a través de una red de ordenadores como Internet

- **Actividad**: Tipo de clase Android que incluye una interfaz gráfica para interaccionar con el usuario y captura y maneja los eventos que se hacen sobre ella.

- **Celda**: Área de cobertura estipulada para receptores o transmisores que pertenecen a la misma estación base.

- **Interfaz**: De forma general se define como un conexión física o funcional entre dos sistemas o dispositivos de cualquier tipo dando una comunicación entre distintos niveles. En este proyecto aparece con dos acepciones:

  - Como interfaz Java que indica los métodos que se deben implementar para la comunicación con clases predefinidas.
  - Aspecto visual de la aplicación con la que interacciona el usuario y accede a las diferentes funcionalidades de ésta.

- **Instanciar**: Creación y almacenamiento en memoria de un objeto Java en el que se reservan posiciones de memoria para cada uno de sus atributos o métodos.

- **Clase**: Modelo de datos que incluye otros modelos de datos llamados atributos y que puede incluir métodos que definen acciones.

- **Objeto**: Particularización de la clase en la que se asignan valores concretos a sus atributos.

- **Parsear**: Proceso de analizar una secuencia de símbolos a fin de determinar su estructura gramatical con respecto a una gramática formal dada.

- **Nodo**: Elemento de árbol lógico con información para referenciar a los elementos con los que está conectado.

- **Mapear**: Replicar datos de una forma de estructuración a otra.

- **Diagrama de clases**: Diagrama en el que se muestran las relaciones de herencia, uso, inclusión, etc. entre clases de un programa

- **Diagrama de paquetes**: Diagrama en el que se muestran las diferentes relaciones entre los paquetes de un programa

- ***Array***: Colección ordenada de elementos de un mismo tipo de datos, agrupados de forma consecutiva en memoria.

- ***Layout***: esquema de distribución de los elementos dentro un diseño

- **Null**: Valor por defecto que tiene una variable u objeto Java que aún no ha sido instanciada.

- **Lista expandible**: Lista con dos niveles en la que los elementos del segundo nivel se agrupan en listas dependientes de un elemento del primer nivel. Inicialmente se muestran los elementos de primer nivel y al pulsar sobre uno de ellos se despliega la lista de segundo nivel dependiente.

- **Encapsular**: Insertar datos representados de diferentes maneras en objetos Java

- **Vista**: Cada uno de los elementos que componen una funcionalidad diferenciada en el layout de una interfaz.

- **Jsoup**: Parseador Java para textos HTML

- **Java**: Lenguage de programación orientado a objetos multiplataforma.

1.5.2 Abreviaturas

- **UMTS** *Universal Mobile Telecommunications System*. Sistema de telecomunicaciones móviles de tercera generación.

- **HSPA** *High-Speed Packet Access*. Acceso a paquetes de alta velocidad. Es la combinación de tecnologías posteriores y complementarias a la 3ª generación de sistemas de telecomunicaciones móviles.

- **GPS** *Global Positioning System*. Sistema de posicionamiento global que permite determinar en todo el mundo la posición de un objeto, una persona o un vehículo.

- **LPS** *Location Based Service*. Servicio basado en localización, que busca ofrecer un servicio personalizado a los usuarios basándose en la mayoría de situaciones en información de ubicación geográfica de estos.

- **WPS** *Wifi Positioning Service*. Servicio de posicionamiento a partir de intensidad de señales *WiFi* recibidas.

- **SSID** *Service Set IDentifier*. Identificador de conjunto de servicio. es un nombre incluido en todos los paquetes de una red inalámbrica (*WiFi*) para identificarlos como parte de esa red. El código consiste en un máximo de 32 caracteres que la mayoría de las veces son alfanuméricos (aunque el estándar no lo especifica, así que puede consistir en cualquier carácter). Todos los dispositivos inalámbricos que intentan comunicarse entre sí deben compartir el mismo SSID.

- **MAC** *Media Access Control*. En las redes de computadoras, la dirección MAC (en español "control de acceso al medio") es un identificador de 48 bits (6 bloques hexadecimales) que corresponde de forma única a una tarjeta o dispositivo de red. Se conoce también como dirección física, y es única para cada dispositivo.

- **GSM** *Global System for Mobile*. El sistema global para las comunicaciones móviles es un sistema estándar de telefonía móvil digital. GSM se considera, por su velocidad de transmisión y otras características, un estándar de segunda generación.

- **GPRS** *General Packet Radio Service*. El servicio general de paquetes vía radio creado en la década de los 80 es una extensión del Sistema Global para Comunicaciones Móviles (GSM) para la transmisión de datos mediante conmutación de paquetes.

- **BTS** *Base Transceiver Station*. Estación base de la red de telefonía móvil que emite y recibe las señales de radio con las que se comunica con los dispositivos situados en su celda.

- **API** *Application Programming Interface*. Conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas.

- **SDK** *Software Development Kit*. Kit de desarrollo de software.

- **REST** *Representational State Transfer*. Familia de arquitecturas para servicios hipermedia distribuidos basados en peticiones HTTP.

- **HTTP** *Hypertext Transfer Protocol*. El protocolo de transferencia de hipertexto es un protocolo de transmisión de datos utilizado en la World Wide Web.

- **XML** *Extensible Markup Language*. Siglas en inglés ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el *World Wide Web Consortium* (W3C) utilizado para almacenar datos en forma legible.

- **JSON** *JavaScript Object Notation*. Formato ligero para el intercambio de datos.

- **HTTPS** *Hypertext Transfer Protocol Secure*. Protocolo seguro de transferencia de hipertexto basado en el protocolo HTTP, destinado a la transferencia segura de datos de Hipertexto.

- **UTC** *Current Local Time*. Hora local actual, de un compromiso entre la versión en inglés: Coordinated Universal Time y la versión en francés: *Temps Universel Coordonné*

- **URL** *Uniform Resource Locator*. Localizador de recursos uniforme es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación, como por ejemplo documentos textuales, imágenes, vídeos, presentaciones digitales, etc.

- **DOM** *Document Object Model*. Modelo de Objetos del Documento o Modelo en Objetos para la Representación de Documentos, es esencialmente una interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos

- **SOAP** *Simple Object Access Protocol*. Es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

- **WSDL** *Web Services Description Language*. Formato XML que se utiliza para describir servicios Web

- **JDK** *Java Development Kit*. Es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red.

- **IDE** *Integrated Development Environment*. Es un programa informático compuesto por un conjunto de herramientas de programación.

- **ADT** *Android Developer Tools*. Herramientas para desarrolladores Android.

- **SHA1** *Secure Hash Algorithm*. Familia de sistemas de funciones hash criptográficas relacionadas de la Agencia de Seguridad Nacional de los Estados Unidos y publicadas por el *National Institute of Standards and Technology* (NIST).

- **AVD** *Android Virtual Device*. Dispositivo virtual Android para emular el funcionamiento de un dispositivo móvil en un ordenador y poder así depurar aplicaciones.

1.6 Contenido de la memoria

De aquí en adelante la memoria se estructura de la siguiente manera:

- Capítulo 2: En este capítulo se exploran los diferentes componentes que permiten crear la aplicación de venta de entradas para eventos cercanos al usuario. Entre ellos se encuentran los diferentes sistemas de geolocalización; las librerías que

permiten la creación y manipulación de mapas; posibles fuentes de información sobre eventos en la web.

- Capítulo 3: En este apartado se pretende proporcionar una visión global de la aplicación mediante una explicación en profundidad de la arquitectura y funcionalidad de la misma. Con este fin, se exponen los elementos que intervienen en el sistema y la forma que tienen de interactuar entre ellos para conseguir la funcionalidad requerida. Además, se especifican los requisitos funcionales y no funcionales de la aplicación y los casos de uso.

- Capítulo 4: En este capítulo se explica de forma detallada las clases que componen la aplicación y su implementación para alcanzar los requisitos funcionales. Además se hace un análisis de la interfaz de cada una de las actividades y se muestra el diagrama de clases para la funcionalidad de cada una.

- Capítulo 5: En este capítulo se presentan las pruebas de funcionalidad que se han realizado sobre la aplicación. Se dividirán las pruebas según las diferentes situaciones que se pueden dar en la aplicación y se comprobará la correcta navegación entre las distintas actividades, el funcionamiento de las rutinas de obtención de datos y la presentación de los datos al usuario.

- Capítulo 6: En este capítulo se exponen las principales conclusiones del proyecto y se enumeran una serie de posibles líneas de trabajo futuras.

Además el proyecto cuenta con los siguientes anexos:

- Apéndice A: En este apéndice se elabora el presupuesto del proyecto.

- Apéndice B: Información sobre la utilización del API de Google Maps para Android.

- Apéndice C: Una introducción a los servicios web y a las arquitecturas y formatos de transmisión de información más utilizados.

- Apéndice D: Instalación y configuración del entorno de trabajo necesarios para desarrollar la aplicación.

- Apéndice E: En este apéndice se ofrece una pequeña guía de usuario para utilizar la aplicación.

Capítulo 2. Estado del arte

2.1. Introducción

En este capítulo se van a explorar los diferentes componentes que permiten crear una aplicación de compra de entradas para eventos adaptativa según contexto para el sistema operativo Android.

En primer lugar se hará una pequeña introducción sobre este sistema operativo y sobre los servicios basados en localización como el desarrollado en este proyecto. A continuación, se explorarán las posibilidades del terminal Android para geolocalizar al usuario y para representar los datos en mapas. Después se inspeccionarán las posibles fuentes de información disponibles en la web.

Las secciones que se van a estudiar son:

- Geolocalización con el paquete android.location de Android.
- Representación de mapas a través de la librería com.google.android.maps.
- Servicio web Google Places.
- Servicio web Google Directions.
- Servicio web Kulturklik.
- Servicio web Movies de iGoogle.
- Página web google.es/movies.

2.2 Android

Android es un sistema operativo de código abierto basado en Linux, que es un núcleo de sistema operativo libre, gratuito y multiplataforma. Está diseñado principalmente para dispositivos móviles con pantalla táctil como teléfonos inteligentes o tabletas, aunque se utiliza también en otros como reproductores MP3, relojes, *netbooks* o incluso televisores. En un inicio fue desarrollado por la empresa Android Inc. que posteriormente fue comprada por Google en el año 2005. Posteriormente, en octubre de 2008, apareció el

HTC Dream, el primer móvil con sistema operativo Android y cinco años después este tipo de dispositivos abarcan un 64% del mercado de telefonía móvil mundial [6] [7].

El sistema operativo proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla en un lenguaje de programación muy conocido como es Java. Esta sencillez junto a la existencia de herramientas de programación gratuitas hacen que una de las cosas más importantes de este sistema operativo sea la cantidad de aplicaciones disponibles, que extienden casi sin límites la experiencia del usuario.

En este sentido, se puede decir que el éxito de este sistema operativo radica en que es completamente libre. Es decir, cualquiera puede obtener el código fuente, inspeccionarlo, compilarlo y modificarlo para adaptarlo a sus preferencias. Esto supone una ventaja para los usuarios, ya que se ha creado una gran comunidad de desarrolladores capaces de identificar los fallos que pueda tener el sistema operativo y subsanarlos. A su vez también supone una gran ventaja para los fabricantes de terminales, ya que pueden hacer modificaciones para adaptar el sistema a las características del terminal y conseguir así mejores prestaciones.

2.3 Servicios basados en Localización (LBS)

Actualmente el uso de teléfonos móviles inteligentes (*smartphones*) ha revolucionado el mundo de las comunicaciones y ha permitido el acceso a Internet desde cualquier situación espacio-temporal. Esto ha dado lugar a una serie de servicios que están basados en la posición actual del usuario y que le proporcionan información sobre su alrededor. Estos servicios se llaman servicios basados en localización, conocidos también por sus siglas en inglés LBS (*Location Based Service*).

Un sistema LBS se basa en la combinación de varios elementos [8]:

- Dispositivo móvil: El elemento con el que interaccionará el usuario y a través del cual se pedirá la información necesaria. Dicho dispositivo móvil puede consistir en un teléfono móvil, un ordenador portátil, una tableta o incluso un navegador de un coche.
- Red de comunicaciones: Es necesario que el dispositivo móvil esté conectado a una red para transferir la información del usuario y su petición a un proveedor de servicios.
- Componente de posicionamiento: Aporta la información de posicionamiento del usuario. Ésta puede obtenerse a través de diferentes técnicas, las más comunes son el sistema GPS, la red de comunicaciones móvil a la que está conectado el dispositivo (*WiFi*) o a través de las torres de telefonía móvil.

- Proveedor de servicios o contenidos: Ofrece servicios al usuario y es el responsable de procesar la petición hecha por éste.

El funcionamiento típico de estos servicios es el siguiente [8]:

1. El usuario, mediante su dispositivo móvil, pulsa la opción necesaria para obtener los lugares cercanos correspondientes.
2. El dispositivo es ubicado por uno de los sistemas de geolocalización. Tras ello, el terminal envía la petición, que contiene la información que se quiere obtener así como la posición calculada.
3. La puerta de enlace (*gateway*) es la encargada de intercambiar mensajes entre la red de comunicaciones móviles e Internet. Dispone de las direcciones web de diferentes servidores de aplicaciones y envía la petición al servidor correspondiente. La puerta de enlace almacenará así mismo cierta información sobre el dispositivo que ha realizado la petición.
4. El servidor de aplicaciones lee la petición y activa el servicio correspondiente.
5. En este punto, el servicio analiza de nuevo la petición y decide qué información adicional necesita tal solicitud.
6. Una vez se dispone de toda la información, el servicio hará una petición para obtener la información solicitada. Una vez terminado el cálculo y obtenida dicha información, ésta se envía de vuelta al usuario a través de Internet, la puerta de enlace y la red de comunicaciones móviles.

2.4. Geolocalización con el paquete android.location de Android

El paquete de Android android.location es el que se encarga de proporcionar las funcionalidades que permiten obtener la geolocalización del dispositivo móvil ya sea utilizando el servicio GPS o la información proporcionada por la red. Antes de analizar el paquete en sí, se ofrecerá una introducción a los sistemas que utiliza el dispositivo.

2.4.1 Introducción a las tecnologías disponibles de geolocalización

En los dispositivos Android, se pueden utilizar tres sistemas de geolocalización distintos. En el dispositivo se diferencian en dos grupos [9]:

- fine_location, para las aplicaciones que necesiten de una mayor precisión a costa de mayores requisitos de memoria y batería. En este grupo estaría el sistema GPS
- coarse_location, para las aplicaciones que pueden prescindir de una mayor precisión para disminuir el consumo de recursos del terminal. En este grupo estarían los sistemas de WPS y *Cell-ID*.

2.4.1.1 GPS (*Global Positioning Service*)

Las siglas GPS significan *Global Positioning System*, sistema de posicionamiento global en castellano. El GPS es un sistema global de navegación por satélite que permite determinar en todo el mundo la posición de un objeto, una persona o un vehículo con una precisión hasta de centímetros (si se utiliza GPS diferencial), aunque lo habitual son unos pocos metros de precisión. El sistema fue desarrollado, instalado y actualmente es operado por el Departamento de Defensa de los Estados Unidos [10].

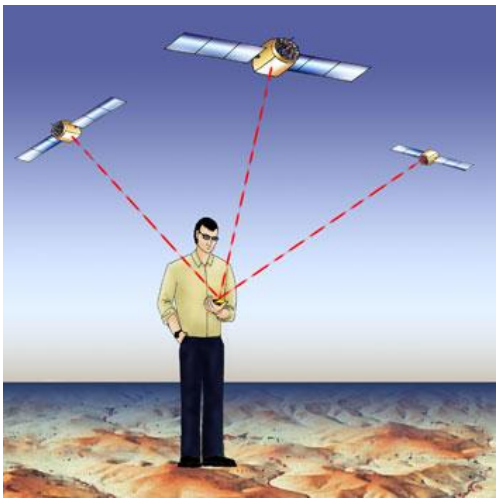

Figura 2.1 Usuario obteniendo su localización por satélites © [11]

Su funcionamiento se debe a una red de 24 satélites en órbita sobre el planeta Tierra, a 20.200 km, con trayectorias sincronizadas para cubrir toda la superficie. Para determinar la posición se utiliza un receptor que localiza automáticamente como mínimo tres satélites de la red. Estos satélites envían continuamente señales con la información relativa a su identificación, posición, hora del reloj de cada uno de ellos, información *doppler*, etc. Con base en estas señales, el aparato sincroniza el reloj del GPS y calcula el tiempo que tardan en llegar las señales al equipo. Mediante el método de "triangulación" (método de trilateración inversa), el receptor mide la distancia al satélite, la cual se basa en determinar la distancia de cada satélite respecto al punto de medición. Conocidas las distancias, se determina la propia posición respecto a los tres satélites. Utilizando la ubicación de cada uno de ellos se obtiene la posición absoluta o las coordenadas reales del punto de medición. Teniendo información de un cuarto satélite, se elimina la falta de sincronización entre los relojes de los receptores GPS y los relojes de los satélites y se puede determinar una posición 3D exacta (latitud, longitud y altitud).

La posición calculada por el receptor GPS requiere en el instante actual, la posición del satélite y el retraso medido de la señal recibida. Estos factores condicionan la precisión del sistema. El receptor compara una serie de *bits* recibida del satélite con una versión interna. Existe una variable de un 1% de un tiempo *bit* en el cálculo del tiempo de llegada de la señal satelital que supone un error de unos 3 metros. Este es el error mínimo posible. A este error mínimo se le añaden otros errores que pueden aparecer como consecuencia de otros factores como son el retraso de la señal en la ionosfera y la troposfera; señal multirruta, producida por el rebote de la señal en edificios y montañas cercanos; errores de los datos de la órbita del satélite; número de satélites visibles; geometría de los satélites visibles o errores locales en el reloj del GPS.

2.4.1.2 WPS (*Wifi Positioning Service*)

El problema del posicionamiento mediante GPS radica en que, a pesar de ser el método más preciso, la señal empeora significativamente en interiores y en aquellas zonas urbanas con muchos y grandes edificios (lo que aumenta el multitrayecto y por tanto el desvanecimiento de la señal - *fading*). Una alternativa en este tipo de escenarios es el posicionamiento mediante puntos de acceso *WiFi* (*hotspots*) [12].

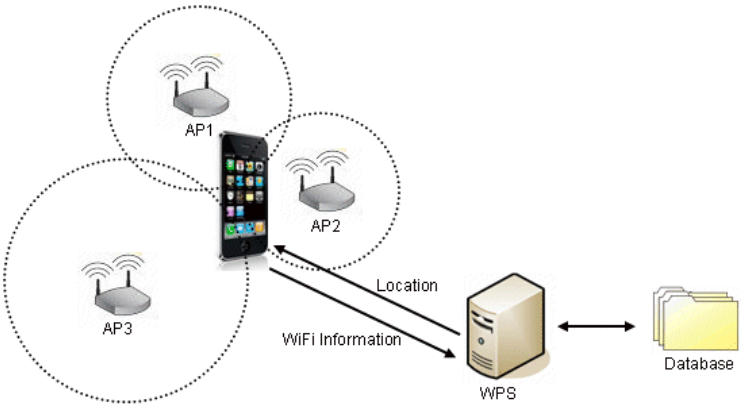

Figura 2.2 Localización a través de WPS conectándose a servidor © [13]

El WPS es un sistema de geolocalización que consiste en un software que obtiene una posición aproximada a partir de los puntos de acceso *WiFi* existentes. Por tanto, en zonas con numerosos puntos de acceso, el terminal puede hacer la triangulación satisfactoriamente de la misma manera que ocurre con la tecnología GPS. La posición de los accesos *WiFi* se conoce por una base de datos y la distancia del usuario al punto de acceso se determina por la intensidad de la señal recibida por éste.

La base de datos que sustenta el funcionamiento del sistema es mantenida por Google. Cuando un teléfono móvil *smartphone* está conectado a una red *WiFi*, el servicio de localización de Android comprueba periódicamente la posición del terminal usando GPS, WPS o *Cell-ID*. Una vez conseguida la ubicación se envían los datos del SSID y MAC del punto de acceso que se está utilizando. Esta información se introduce en una base de datos junto con la posición del terminal. Cuando se quiere utilizar el sistema, se

comprueban los puntos de acceso *WiFi* que hay alrededor del usuario, se consulta la base de datos y se realiza la triangulación [14].

Aunque no se obtiene tan buena precisión como con tecnología GPS (aproximadamente 20 m), tiene la ventaja de no necesitar visión directa como en el caso anterior, además de ser un método mucho más económico en términos de consumo de recursos del terminal. Destacar la nula aplicación en zonas rurales donde la ausencia de puntos de acceso imposibilita el despliegue de dicha tecnología.

2.4.1.3 Cell-ID

Cell-ID es el método de localización menos preciso y más sencillo disponible en un *smartphone*. Se basa en obtener la posición del usuario basándose en la celda de la red de telefonía móvil a la que está conectado. Este sistema está operativo en la mayoría de tipos de red incluyendo redes GSM, GPRS y UMTS/HSDPA.

Para que el sistema esté operativo, es necesario que la red sea capaz de identificar a la estación base o BTS a la cual está conectada el dispositivo móvil y la ubicación de ésta. Estos datos se proporcionan al servidor de localización que se los facilita al servicio de localización de la aplicación [15].

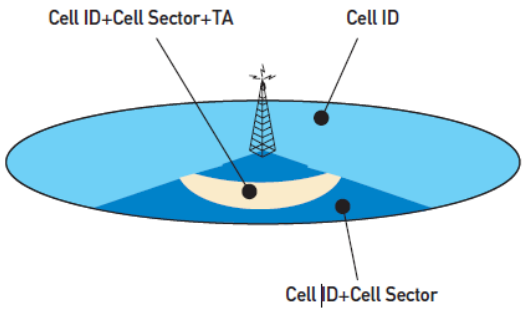

Figura 2.3 Diferentes áreas de precisión dependiendo del tipo de localización de celda © [16]

Dado que el usuario se puede encontrar en cualquier punto de la celda, la precisión del sistema depende del tamaño de la misma. Por ello en la mayoría de los casos será una aproximación bastante burda y que variará dependiendo del tipo de celda en la que se encuentre conectado el terminal y de si se encuentra en un medio rural o urbano.

Con respecto al tipo de red al que pertenezca la celda vamos a tener tamaños de celda diferentes abarcando más terreno las celdas de GSM, después las de GPRS y por último las más pequeñas serían las de UMTS/HSDPA. Se puede mejorar la precisión incluyendo una medida del TA (*Time Advance*) en redes GSM/GPRS o de RTT (*Round-Trip Time*) en redes UMTS/HSDPA, que calculan el retardo que existe entre las señales que envía el dispositivo a la estación base. Con el retardo se puede obtener la distancia del terminal a la estación y reducir la imprecisión en la ubicación del usuario.

En el medio urbano existe una mayor densidad de BTS's y por lo tanto el tamaño de celda es menor y la precisión del sistema mayor. En cambio en el medio rural una misma estación base cubre un territorio más amplio y por lo tanto la precisión será menor.

2.4.2 Paquete `android.location`

La librería de android.location ofrece las clases e interfaces necesarias para utilizar los sistemas de geolocalización, que nos permite el dispositivo móvil con sistema operativo Android [9].

Para la utilización de estos servicios en una aplicación Android se necesita de una serie de permisos que se declararán en el AdroidManifest. El API de Android permite dos formas distintas para conseguir la ubicación del usuario:

- Mediante el GPS: Se utiliza el GPS para aplicaciones en las que se necesitan una mayor precisión en los datos. En este caso el permiso que se incluye es el ACCESS_FINE_LOCATION.
- Mediante información de red: Se utiliza la información de localización suministrada por el sistema WPS o por el sistema *Cell-ID*. La ubicación en este caso es bastante más burda que para el caso del GPS aunque el gasto de batería y recursos del terminal es menor. En este caso se debe incluir el permiso ACCESS_COARSE_LOCATION.

El permiso ACCESS_FINE_LOCATION también incluye el permiso ACCESS_COARSE_LOCATION al ser este último menos restrictivo que el primero.

La clase principal a través de la cual se accede a los servicios de localización en Android es LocationManager. Las principales acciones que permite esta clase son las de obtener la ubicación exacta del usuario, actualizar esta ubicación de forma periódica o disparar una aplicación cuando el usuario se encuentre en las proximidades de un determinado punto geográfico.

No se puede instanciar directamente la clase LocationManager. Para ello se realiza una llamada al método getSystemService() de la interfaz Contex que pertenece al paquete android.content. Esta clase abstracta permite acceder a información global sobre las aplicaciones del dispositivo (permite el acceso a recursos, lanzar aplicaciones, emitir y recibir operaciones de otras aplicaciones, etc.). Su implementación viene dada por el sistema operativo Android en el cual estamos ejecutando nuestra aplicación.

El objeto LocationManager se deberá instanciar por lo tanto de esta manera:

```
LocationManagerlocationManager =
    (LocationManager)this.getSystemService(Context.LOCATION_SERVICE);
```

Una vez que se tenga un objeto LocationManager, se necesita un LocationProvider que es la referencia al sistema de localización que se utiliza. Se instancia a través del objeto LocationManager de la aplicación:

```
LocationProvider provider =
    locationManager.getProvider(LocationManager.GPS_PROVIDER);
```

En este ejemplo se utiliza el servicio GPS como servicio proveedor de la localización. Si se desea una ubicación menos costosa y menos precisa se puede hacer uso de la información de red, en cuyo caso utilizaríamos las constante NETWORK_PROVIDER en lugar de GPS_PROVIDER.

Aparte de estos dos tipos de proveedor existe uno más al que podemos referenciar por la constante PASSIVE_PROVIDER. Hace alusión a un proveedor externo perteneciente a otras aplicaciones que ya se están sirviendo de los servidores de localización para calcular la situación del usuario. En este caso en vez de hacerse una petición explícita a los proveedores activos se obtienen los datos a través de una aplicación que actúa de intermediaria.

La clase LocationProvider incluye una gran cantidad de métodos que permiten conocer las características del proveedor que se está utilizando. Se puede obtener información sobre su precisión o los requisitos de potencia (getAccuracy(), getPowerRequirement()), comprobar si su uso tiene asociado un coste (hasMonetaryCost()), si devuelve información sobre la altitud o la velocidad del usuario (supportsAltitude(), supportsSpeed()) o la orientación del dispositivo (supportsBearing()).

Toda esta información se puede incluir sobre la clase Criteria en la cual se describen las propiedades cualitativas (por ejemplo: isAltitudeRequired()) y cuantitativas (por ejemplo: setAltitudeRequired()) que se requieren de un proveedor de localización para que el sistema devuelva el que mejor se adapte a las necesidades de la aplicación entre los que estén disponibles:

```
LocationProvider.meetsCriteria(Criteria criterio)
```

Una vez instanciado el proveedor, existen diferentes métodos de la clase LocationManager para obtener la posición del usuario. La forma más sencilla para hacerlo es utilizar la clase LocationListener que es la que se encarga de recibir las notificaciones por parte del LocationManager cuando hay cambios en la localización. Para instanciarla se debe utilizar el método requestLocationUpdates de la clase LocationManager que tiene la siguiente estructura:

```
requestLocationUpdates (String provider, long minTime, float
  minDistance, LocationListener  listener)
```

De esta forma se configura la periodicidad con la que el proveedor devuelve la situación del usuario. Se puede configurar para obtener una nueva ubicación cada vez que se detecte un desplazamiento superior a una cierta distancia (floatminDistance) y/o que haya pasado un período de tiempo definido (longminTime).

Los métodos del LocationListener son abstractos, por lo que deberán ser definidos por el programador para implementar la funcionalidad que se desee para cada aplicación. Los principales son:

- onLocationChange(Locationlocation): Se ejecuta cuando se produce un cambio en la localización
- onProviderDisabled(Stringprovider): Se ejecuta cuando se ejecuta el comando de actualización y el proveedor está desactivado.
- onProviderEnabled(Stringprovider): Se ejecuta cuando el usuario habilita el proveedor

Para detener las actualizaciones periódicas de la ubicación llamamos al método LocationManager.removeUpdates (Listenerlistener).

También es posible obtener un único emplazamiento del usuario con el método requestSingleUpdate() u obtener el último emplazamiento conocido con getLastKnownLocation(), ambos de la clase LocationManager.

La ubicación del usuario se devuelve como un objeto de la clase Location. En esta clase se almacenan los valores de latitud, longitud y fecha de la medición y otros valores opcionales como son la altitud, velocidad y la orientación.

Normalmente el usuario no desea conocer su emplazamiento en referencia a una latitud y longitud, sino que se espera una dirección, calle, etc. Para transformar una posición expresada en grados en una dirección real, se utiliza la clase Geocoder que internamente se conecta con un API dependiente de un servicio web. Para ello podemos usar el método:

```
getFromLocation (double latitude, double longitude, int maxResults)
```

También existe la posibilidad de hacer la geocodificación inversa, a partir de una dirección dada en formato de texto obtener las coordenadas de la posición. El formato de la dirección debe ser el mismo que se utiliza en cada país como formato postal, evitando añadir nombres de edificios o datos adicionales a la dirección. El método a utilizar sería:

```
getFromLocationName (String locationName, int maxResults)
```

2.5 Representación en el mapa

En el estudio tecnológico y el inicio del proyecto se utilizó la versión 1 del API de Google Maps para Android. Después, ante proyección de obsolescencia de esta API y su substitución por la versión 2, se modificó el proyecto para que se adaptara a estos cambios. En este apartado se va a hacer un repaso sobre la segunda versión y en el anexo A se recogerá la primera versión.

2.5.1 Google Maps Android v2

La nueva versión del API de Google Maps para Android incluye una serie de mejoras frente a la primera versión [17]:

- El API pasa a distribuirse como parte del SDK de Google Play Services y es necesario que el usuario lo tenga instalado en su dispositivo y que disponga de una cuenta de Google para poder utilizarlo.
- Los mapas se encapsulan dentro de la clase MapFragment que hereda de la clase Fragment de Android. Al transformar el mapa en un fragmento puede ocupar toda la pantalla, como suele ocurrir en un dispositivo móvil, o se puede combinar con otras funcionalidades en pantallas más grandes como la de una tableta.
- El mapa al estar definido en un Fragment puede extender de la clase Activity en lugar de la clase MapActivity.
- Los Mapas pasan a tener una representación por vectores que necesitan una menor cantidad de datos para representarlos. Esto hace que la aparición del mapa en el dispositivo y la interacción con él por parte del usuario sea más rápida y gaste un menor ancho de banda.
- La memoria caché se ha perfeccionado y por lo tanto los mapas se muestran usualmente sin áreas en blanco.
- Los mapas se representan en 3D. El usuario puede mover el punto de mira y ver el mapa con perspectiva.

La diferencia principal al desarrollar la aplicación con la nueva versión del API será el componente que se utiliza para la inclusión de mapas en la aplicación. En la anterior versión, para incluir un mapa en la aplicación se utilizaba un control de tipo MapView, que además requería que su actividad contenedora fuera del tipo MapActivity. La nueva API deja de utilizar estos dos componentes y se pasa a tener sólo uno, un nuevo tipo específico de Fragment llamado MapFragment. Esto nos permitirá entre otras cosas añadir uno (o varios, esto también es una novedad) mapas a cualquier actividad, sea del tipo que sea, y contando por supuesto con todas las ventajas del uso de Fragment's.

Dado que el nuevo control de mapas se basa en Fragment's, si se quiere mantener la compatibilidad con versiones de Android anteriores a la 3.0 se ha de utilizar la librería de soporte android-support.

Para poder utilizar el API de Google Maps es necesario obtener una API Key y configurar adecuadamente el AndroidManifest. Se puede consultar cómo hacerlo en el apartado IV de la parte B del anexo.

Una vez terminada la configuración de todo lo necesario se puede comenzar con la parte del código. Para mostrar un mapa con la funcionalidad básica sólo es necesario añadir en un componente del tipo Fragment que dependa de la clase com.google.android.gms.maps.SupportMapFragment en el *layout* de la actividad principal. Al utilizar el tipo Fragment, la actividad principal deberá extender de FragmentActivity en lugar de Activity.

Sobre el mapa se pueden hacer varias acciones para representar los datos que se desean. En la anterior versión de la API de Google Maps el acceso y modificación de los diferentes datos del mapa se hacían sobre diferentes componentes, lo que complicaba el desarrollo de la aplicación al tener que tratar por un lado con la imagen del mapa (MapView), por otro lado con el controlador (MapController), etc. Con la nueva API, todas las operaciones se realizarán directamente sobre un objeto GoogleMap, el componente base de la API. Accederemos a este componente llamando al método getMap() del fragmento MapFragment que contenga el mapa [18]:

```
GoogleMap mapa = ((SupportMapFragment) getSupportFragmentManager()
                          .findFragmentById(R.id.map)).getMap();
```

Sobre este objeto se hacen todas las acciones que afectan al mapa.

Para modificar el tipo de mapa que queremos mostrar sólo hay que llamar al método setType() de la clase GoogleMap pasándole como parámetro el tipo de mapa:

- MAP_TYPE_NORMAL: Mapa normal de carreteras con el nombre de ciudades, calles y carreteras.
- MAP_TYPE_HYBRID: Fotografía desde satélite con mapa de carreteras sobreimpreso y con los nombres de ciudades, calles y carreteras.
- MAP_TYPE_SATELLITE: Fotografía desde satélite
- MAP_TYPE_TERRAIN: Mapa que incluye colores, líneas de contorno y etiquetas con información topográfica.

En cuanto al movimiento sobre el mapa, con esta nueva versión de la API permite una mayor libertad ya que se puede mover libremente la cámara por un espacio 3D. De esta forma, ya no sólo podremos hablar de latitud-longitud (*target*) y *zoom*, sino también de orientación (*bearing*) y ángulo de visión (*tilt*).

El movimiento de la cámara se realiza llamando a los métodos moveCamera() o animateCamera() de nuestro objeto GoogleMap, dependiendo de si queremos que la actualización de la vista se muestre directamente o de forma animada, respectivamente. Como parámetro de estos métodos se utiliza un objeto CameraUpdate con los parámetros necesarios.

Para los movimientos más básicos, como la actualización de la latitud y longitud o el nivel de zoom, se puede utilizar la clase CameraUpdateFactory y sus métodos estáticos. Así por ejemplo, para cambiar sólo el nivel de zoom se puede utilizar los siguientes métodos para crear el CameraUpdate deseado:

- CameraUpdateFactory.zoomIn(): Aumenta en 1 el nivel de zoom.
- CameraUpdateFactory.zoomOut(): Disminuye en 1 el nivel de zoom.
- CameraUpdateFactory.zoomTo(nivel_de_zoom): Establece el nivel de zoom.

Por su parte, para actualizar sólo la latitud-longitud de la cámara se utiliza:

- CameraUpdateFactory.newLatLng(lat, long): Establece las coordenadas latitud y longitud del centro de la pantalla expresadas en grados.

También se pueden modificar los dos parámetros anteriores de forma conjunta con el método siguiente:

- CameraUpdateFactory.newLatLngZoom(lat, long, zoom). Establece las coordenadas latitud y longitud del centro de la pantalla y el zoom.

Para moverse lateralmente por el mapa (*panning*) se puede utilizar los métodos de *scroll*:

- CameraUpdateFactory.scrollBy(scrollHorizontal, scrollVertical). *Scroll* expresado en píxeles.

Para modificar todos los parámetros de la cámara o varios de ellos simultáneamente se dispone del método general CameraUpdateFactory.newCameraPosition() que recibe como parámetro un objeto de tipo CameraPosition. Este objeto se construye indicando todos los parámetros de la posición de la cámara a través de su método Builder().

Ejemplo (Figura 2.4): vista del mapa centrada sobre el parque de El Retiro de Madrid, con un nivel de zoom de 19, una orientación de 45 grados para que el noreste esté hacia arriba y un ángulo de visión de 70 grados de forma que veamos en 3D el monumento a Alfonso XII en la vista de mapa NORMAL:

```
LatLng madrid = new LatLng(40.417325, -3.683081);
CameraPosition camPos = new CameraPosition.Builder()
        .target(madrid)   //Centramos el mapa en Madrid
        .zoom(19)         //Establecemos el zoom en 19
```

```
        .bearing(45)      //Establecemos la orientación con el noreste
arriba
        .tilt(70)         //Bajamos el punto de vista de la cámara 70
grados
        .build();

CameraUpdate camUpd3 =
    CameraUpdateFactory.newCameraPosition(camPos);

mapa.animateCamera(camUpd3);
```

Figura 2.4 Imagen de El Retiro de Madrid obtenida de Google Maps © [17]

Los eventos sobre el mapa también se manejan desde el propio objeto mapa, a diferencia de con la versión anterior de la API que se debía crear una nueva capa (Overlay) para capturar los eventos principales de pulsación. Sin embargo, el nuevo componente de mapas soporta directamente los eventos de *click*, *click* largo y movimiento de cámara, y se implementan mediante su método set correspondiente de la clase GoogleMap. Los métodos reciben como parámetros, en forma de objeto LatLng, las coordenadas de latitud y longitud sobre las que ha pulsado el usuario [19].

- setOnMapClickListener() : Evento de pulsación normal en la pantalla. Se sobrescribe el método onMapClick(LatLng point).
- setOnMapLongClickListener(): Evento de pulsación larga en la pantalla. Se sobrescribe el método onLongMapClick(LatLng point).

También se puede capturar el evento de cambio de cámara mediante el método setOnCameraChangeListener sobrescribiendo el método onCameraChange(). Este método recibirá como parámetro un objeto CameraPosition con las características del movimiento que ha hecho el usuario en la pantalla.

La siguiente utilidad importante de los mapas es la posibilidad de incluir marcadores para destacar ubicaciones concretas. En la API v1, se necesitaba añadir una nueva capa (overlay) al mapa y dibujando nuestro marcador como parte de su evento draw(). En la nueva versión de la API tendemos toda esta funcionalidad integrada en la propia vista de

mapa, y agregar un marcador simplifica llamando únicamente al método addMarker() pasándole como parámetro la posición en forma de objeto LatLng y el texto a mostrar en la ventana de información del marcador.

Ejemplo (Figura 2.5): añadiremos un menú de forma que cuando lo pulsemos se añada automáticamente un marcador sobre España con el texto "País: España". Veamos cómo escribir un método auxiliar que nos ayuda a hacer esto pasándole las coordenadas de latitud y longitud:

```
private void mostrarMarcador(double lat, double lng)
{
    mapa.addMarker(new MarkerOptions()
        .position(new LatLng(lat, lng))
        .title("Pais: España"));
}
```

Para capturar el evento de pulsación sobre un marcador se asigna al mapa mediante el método setOnMarkerClickListener() y se sobrescribe el método onMarkerClick(). Dicho método recibe como parámetro el objeto Marker pulsado, de forma que podamos identificarlo accediendo a su información (posición, título, texto, etc.). De la misma manera podemos controlar el evento de pulsar sobre la ventana de información del marcador con el método setOnInfoWindowClickListener() del mapa sobrescribiendo el método onInfoWindowClick() que, al igual que en el caso anterior, tiene como parámetro el objeto Marker correspondiente al objeto pulsado que contendrá la información necesaria para identificarlo.

Por último, otra utilidad es la de dibujar líneas y polígonos sobre el mapa. Estos elementos son muy utilizados para trazar rutas o delimitar zonas del mapa. Para realizar esto en la versión 2 de la API se actúa, una vez más, directamente sobre la vista de mapa, sin necesidad de añadir *overlays* o similares. Los objetos que se van a utilizar son PolylineOptions para mostrar líneas y PolygonOptions para mostrar polígonos.

Para dibujar una línea primero se crea un nuevo objeto PolylineOptions sobre el que se añade, utilizando su método add(), las coordenadas (latitud-longitud) de todos los puntos que conformen la línea. Tras esto se establece el grosor y color de la línea llamando a los métodos width() y color() respectivamente, y por último se añade la línea al mapa mediante su método addPolyline() pasándole el objeto PolylineOptions recién creado.

El funcionamiento para crear un polígono es muy similar. Se crea un nuevo objeto PolygonOptions y se añaden las coordenadas de sus vértices en el sentido de las agujas del reloj. El ancho y color de la línea se establecen mediante los métodos strokeWidth() y strokeColor(). El dibujo final del polígono sobre el mapa se añade mediante el método addPolygon().

Un ejemplo de representación de líneas en el mapa sería (Figura 2.6):

```
private void mostrarLineas()
{
```

```
    //Dibujo con Lineas

    PolylineOptions lineas = new PolylineOptions()
            .add(new LatLng(45.0, -12.0))
            .add(new LatLng(45.0, 5.0))
            .add(new LatLng(34.5, 5.0))
            .add(new LatLng(34.5, -12.0))
            .add(new LatLng(45.0, -12.0));

    lineas.width(8);
    lineas.color(Color.RED);

    mapa.addPolyline(lineas);
}
```

Figura 2.5 Representas marcador en mapa © [18]

Figura 2.6 Representar líneas en mapa © [19]

2.6 Servicios Web

Para obtener datos de una manera sencilla y rápida de la red se utilizan los servicios web. En este apartado se va a hacer un estudio de aquellos que contienen información interesante para la aplicación que se quiere desarrollar. Para más información acerca de qué es un servicio web y los protocolos y formatos de transferencia de datos que se utilizan, consultar el anexo C.

2.6.1 Localización de lugares con Google Places:

El servicio web de Google Places permite explorar los elementos que se encuentran alrededor de la posición del usuario. Podemos obtener información sobre establecimientos, ubicaciones geográficas o lugares de interés. Este servicio web es un servicio REST y por lo tanto funciona mediante solicitudes HTTP [20].

Principalmente Google Places ofrece cuatro tipos de solicitudes básicas:

- Búsqueda de lugares: Lista de lugares cercanos en función de la ubicación del usuario.
- Solicitudes de detalles de lugar: Devuelve información más detallada sobre un lugar concreto.
- Visitas de lugar: Permite registrar las visitas de un usuario a un lugar. Mediante este mecanismo se puede medir la popularidad de un lugar y dar prioridad a unos resultados sobre otros en la aplicación
- Informes de lugar: Permite añadir lugares nuevos al servicio de Google Places y eliminar aquellos que haya añadido tu aplicación.

Para realizar una petición para la búsqueda de lugares se utiliza una URL HTTP con el siguiente formato:

https://maps.googleapis.com/maps/api/place/search/*output*?*parameters*

La etiqueta output indica el formato en el que se va a devolver la respuesta. Se puede elegir entre XML [60] y JSON [62]. La etiqueta parameters incluye una lista de parámetros necesarios para la búsqueda separados por el carácter '&'. Algunos de los parámetros son obligatorios y algunos opcionales. Las opciones de búsqueda serán:

- location (obligatorio): indica la latitud/longitud que se desea utilizar para obtener la información del lugar. Se debe proporcionar como un objeto google.maps.LatLng, es decir, la Latitud se representará en grados entre -90º y 90º y la Longitud entre -180º y 180º. En la Latitud, si se dan valores fuera de los límites, se aproxima al extremo más cercano (una latitud de 100º se tomaría como 90º).En la Longitud si se dan valores fuera de los límites se interpretan de una forma circular para expresarlos dentro de ellos (si tenemos una Longitud de 310º se interpretaría como -50º o una de 520º como 160º)
- radius (obligatorio): indica la distancia (en metros) que se desea utilizar para obtener resultados de lugar alrededor de la posición dada.
- types (opcional): restringe los resultados a los lugares que coincidan al menos con uno de los tipos especificados. Los tipos se deben separar con una pleca (type1|type2|etc).
- language (opcional): el código de idioma que indica en qué idioma se deben devolver los resultados, si está disponible.
- name (opcional): término que se debe comparar con los nombres de lugares. Permite obtener únicamente resultados que contengan el valor name especificado.
- sensor (obligatorio): indica si la solicitud de lugar procede de un dispositivo que utiliza un sensor de ubicación (por ejemplo, un GPS) para determinar la localización enviada en la solicitud. Este valor debe ser true o false.
- key (obligatorio): indica la clave de API de la aplicación. Esta clave permite identificar una aplicación al administrar cupos, de forma que los lugares añadidos desde una aplicación estén inmediatamente disponibles para dicha aplicación.

La respuesta a la petición contiene tres campos:

- status: contiene el estado de la solicitud o motivo por el que ha fallado la solicitud de búsqueda de lugar. Indica si se devuelven correctamente los resultados (OK), si la consulta se ha realizado correctamente pero no hay resultados que devolver (ZERO_RESULTS) o si la consulta ha fallado (OVER_QUERY_LIMIT, REQUEST_DENIED, INVALID_REQUEST).
- results: contiene el conjunto de lugares y la información sobre ellos. Google Places API devuelve un total de 20 resultados de establishment como máximo.
- html_attributions: contiene un conjunto de atributos sobre los datos que se mostrarán al usuario.

Los resultados que vienen dentro del campo results se listan ordenados por importancia. Esta importancia viene dada por la cantidad de visitas que tiene cada enlace dentro de la aplicación.

Cada resultado tendrá una serie de atributos encapsulados dentro de los siguientes campos:

- name: contiene el nombre del resultado devuelto.
- vicinity: contiene el nombre de una función de una ubicación cercana. Con frecuencia, esta función se refiere a una calle o a un barrio de los resultados devueltos.
- types: contiene un conjunto de tipos de funciones que describe los resultados devueltos.
- geometry: contiene la información de ubicación del resultado.
- icon: contiene la URL de la imagen del icono mediante el cual se va a representar el resultado en un mapa.
- reference: contiene un *token* único que puede utilizar para recuperar información adicional sobre un lugar al realizar una solicitud de detalles de lugar. No se garantiza que se vaya a mostrar el mismo *token* para un determinado lugar en diferentes búsquedas.
- id: contiene un identificador estable único que representa un lugar. No se puede utilizar para recuperar información sobre el lugar, pero se garantiza que será válido en diferentes sesiones.

Como ejemplo hacemos una petición buscando todos los lugares a 500 metros de la Puerta del Sol que tengan la palabra “teatro” en su nombre o en su descripción:

https://maps.googleapis.com/maps/api/place/search/json?location=40.4165,-3.7037&radius=500&language=es&name=teatro&sensor=false&key=AIzaSyDlqdFUZifGkhtAh5_er8VDmcgmUK_sQKA

La respuesta a esta petición nos devuelve el número máximo de resultados que nos puede proporcionar, 20. Es importante resaltar un elemento devuelto que no aparecía en la información general del API, next_page_token. Repitiendo la misma búsqueda añadiendo el atributo pagetoken e igualándolo al código que se nos devuelve bajo el atributo next_page_token, obtenemos los siguientes 20 resultados de la lista. La petición sería:

https://maps.googleapis.com/maps/api/place/search/json?location=40.4165,-3.7037&radius=500&language=es&name=teatro&sensor=false&key=AIzaSyDlqdFUZifGkhtAh5_er8VDmcgmUK_sQKA&pagetoken=ClRJAAAAOQZT7Oe5qZlb-pLgWtzmxVrFbXU6gNPt4I0cIFXlXHbFBK1I3o7Aar81HC7W3zlrGBPd7fHiy5tx-uNg3cfNZkBfQYb-X97-D_MDEzJWzHYSEOnTL9pm7VCRO97Qw7Yp7A0aFPBZzSO50_fO6DumvQoK2Ka2DcHE

Que devuelve otros 15 resultados más.

Hay que recordar que la posición de los resultados en la lista se organiza por el número de visitas que ha recibido cada resultado. Este número de visitas se personaliza para cada aplicación gracias a la inclusión de la clave API en la petición.

Se han incluido los dos primeros valores, correspondientes al Teatro Albéniz y al Reina Victoria:


```
{
   "html_attributions" : [],
   "next_page_token" : "ClRJAAAAOQZT7Oe5qZlb-
pLgWtzmxVrFbXU6gNPt4I0cIFXlXHbFBK1I3o7Aar81HC7W3zlrGBPd7fHiy5tx-uNg3cfNZkBfQYb-X97-
D_MDEzJWzHYSEOnTL9pm7VCRO97Qw7Yp7A0aFPBZzSO50_fO6DumvQoK2Ka2DcHE",
"results" : [
      {
         "geometry" : {
            "location" : {
               "lat" : 40.4155770,
               "lng" : -3.7039660
}
         },
         "icon" : "http://maps.gstatic.com/mapfiles/place_api/icons/generic_business-
71.png",
         "id" : "4bf0888ba633d168148c82bc17140110c8100b0a",
         "name" : "Teatro Albéniz",
         "reference" : "CnRtAAAAEFqh-
IUfRrN7RoAyZ_emNoPRrEQyRjk8sgLhLM00HbX9tq3qRjtw1n0JMnjUPx81SXzNzXcAPuY80nkKM1QLgAfYmom07Y5
9tKlO6ql6mGWuVg1tOJcpkLovn3e9A22zY9dMsORKkMZQreZWcSs5FxIQp25aIfB1RmMK2nIq6U5z1xoUMYwSooVJz
5fFZ7WDByclWb3KmPc",
         "types" : [ "establishment" ],
         "vicinity" : "Calle de la Paz, 11, Madrid"
      },
      {
         "geometry" : {
            "location" : {
               "lat" : 40.4166170,
               "lng" : -3.7007860
            }
         },
         "icon" : "http://maps.gstatic.com/mapfiles/place_api/icons/generic_business-
71.png",
         "id" : "9a76a70e7a83bf131e510cfa62839a0bcf23038b",
```

```
         "name" : "Reina Victoria",
         "rating" : 4.0,
         "reference" : "CnRrAAAAQEoAer7-oivO4F3_8RIgK-
hmitQhnDGuWPgaOK4ItMfUkk3Tr2NwSmrTEDqnj91vlaVTJUH2msypf7yoanY9qB_Mp4E13epFVpW13Iqe5Le-zLV-
l4yxF8z0NhIees3rddVA0EmacXtI8mH0g8oIbRIQsv9fQOeM4g-HOwF-5S8iLBoUlxxjrQMvzNMOw4TZhJZ0HGm-
7QA",
         "types" : [ "establishment" ],
         "vicinity" : "Carrera de San Jerónimo, 20, 28014, Madrid"
      },
 (…)
],
   "status" : "OK"
}
```


Si se quiere obtener más información sobre uno de los lugares que ha devuelto una búsqueda se puede hacer una "solicitud de detalles de lugar". El formato que va a seguir esta búsqueda va a ser la siguiente:

https://maps.googleapis.com/maps/api/place/details/*output*?*parameters*

La etiqueta output funciona igual que en la petición anterior (JSON, XML).

Los parámetros que se necesitan son:

- reference(obligatorio): un identificador textual del lugar que obtendremos de una búsqueda. Recordemos que no se garantiza que el identificador sea igual para un mismo lugar en búsquedas diferentes.
- language(opcional): el código de idioma que indica en qué idioma se deben devolver los resultados.
- sensor (obligatorio): indica si la solicitud de detalles de lugar procede de un dispositivo que utiliza un sensor de ubicación (por ejemplo, un GPS).
- key (obligatorio): indica la clave de API de la aplicación.

La respuesta a este tipo de petición tiene los mismos campos que en el caso de petición de búsqueda de lugares: status, result y html_attributions.

Los atributos que nos podemos encontrar dentro del campo de resultados son:

- name: nombre del resultado devuelto.
- vicinity: contiene el nombre de una función de una ubicación cercana.
- types: tipo del resultado devuelto.
- formatted_phone_number: contiene un número de teléfono.
- formatted_address: dirección del lugar. Esta dirección suele corresponder con la "dirección postal".
- address_components: Se descompone la dirección anterior en un conjunto de componentes de dirección. Tendremos uno para la calle, el número de portal, la ciudad, etc. Los campos de cada conjunto son:

    - types: es un conjunto que indica el tipo de componente de la dirección.
    - long_name: es la descripción completa o el nombre completo del componente de la dirección tal como lo ha devuelto el *geocoder*.
    - short_name: es un nombre textual abreviado del componente de la dirección (si está disponible).

- geometry: contiene el campo location que lleva los valores de latitud y longitud para un lugar.
- url: contiene la URL de la página de lugar de Google oficial del establecimiento.
- rating: contiene la puntuación global del usuario del establecimiento como se muestra en Google Maps.
- icon: contiene la URL de un icono sugerido que se puede mostrar al usuario en el mapa junto con el resultado.
- reference: contiene un *token* que se puede utilizar para futuras consultas al servicio de detalles.
- id: contiene un identificador estable único que representa un lugar.

Utilizamos la referencia del Teatro Albéniz, que se nos devolvió en la consulta del ejemplo anterior, para enviar una petición que nos devuelva la información sobre el teatro:

https://maps.googleapis.com/maps/api/place/details/json?lenguage=es&sensor=false&key=AIzaSyDlqdFUZifGkhtAh5_er8VDmcgmUK_sQKA&reference=CnRtAAAAEFqh-IUfRrN7RoAyZ_emNoPRrEQyRjk8sgLhLM00HbX9tq3qRjtw1n0JMnjUPx81SXzNzXcAPuY80nkKM1QLgAfYmom07Y59tKlO6ql6mGWuVg1tOJcpkLovn3e9A22zY9dMsORKkMZQreZWcSs5FxIQp25aIfB1RmMK2nIq6U5z1xoUMYwSooVJz5fFZ7WDBycIWb3KmPc

En la respuesta podemos ver la información devuelta sobre el lugar buscado. Entre esta información se incluye una página web del recinto y opiniones de usuarios de google acerca de él:


```
{
   "html_attributions" : [],
   "result" : {
      "address_components" : [
   {"long_name" : "11","short_name" : "11","types" : [ "street_number" ] },
{"long_name" : "Calle de la Paz","short_name" : "Calle de la Paz","types" : [ "route" ] },
{"long_name" : "Madrid","short_name" : "Madrid","types" : [ "locality", "political" ] },
     {"long_name" : "Madrid","short_name" : "Madrid","types" : [
   "administrative_area_level_2", "political" ] }, {"long_name" : "Comunidad de
   Madrid","short_name" : "Comunidad de Madrid","types" : [ "administrative_area_level_1",
   "political" ] },
         {"long_name" : "ES","short_name" : "ES","types" : [ "country", "political" ] },
         {"long_name" : "28012","short_name" : "28012","types" : [ "postal_code" ] }
],
      "formatted_address" : "Calle de la Paz, 11, Madrid, España",
"formatted_phone_number" : "915 31 83 11",
      "geometry" : {"location" : {  "lat" : 40.4155770, "lng" : -3.7039660} },
"icon" : "http://maps.gstatic.com/mapfiles/place_api/icons/generic_business-71.png",
"id" : "4bf0888ba633d168148c82bc17140110c8100b0a",
      "international_phone_number" : "+34 915 31 83 11",
```

```
      "name" : "TeatroAlbéniz",
      "reference" : "CnRtAAAAcbuNmJTNkXeV3wukadsfFwD-
uaMML3qWT8utrVu5V0Om6KwU016hYcw1gl88xM7bLZ0KisGdV3ORNxWfWtZtxxVknHvHFVM9WxfCLBOGNGIV_0QfHE
LIj1TEIaYeFWdDnebI4boUnZLg-
ZYxlAftghIQfkzqdfeJGoBR2WulJngz5RoUvwKclkPSOxFRxRMkXvxISMLhiZo",
      "reviews" : ["aspects" : [{"rating" : 3, "type" : "overall"}],
"author_name" : "Un usuario de Google",
            "text" : "KRIMA.... Ekplisome.....pos ene
dinatonnasimviaftostinmadriti;iparhounlogoipolitikoi;Denmporonatopistepsoeilikrina,Efhomai
na min pragmatopiithiafti i apofasi ton pion; den gnorizo, allaelpizo.D:TOULIO Grecia.",
"time" : 1229622382 },
{ "aspects" : [{"rating" : 0, "type" : "overall"}],
"author_name" : "Un usuario de Google",
            "text" : "ESTÁ CERRADO TAPIADO DE ARRIBA ABAJO.GRACIAS ESPECULADORES",
"time" : 1313244210
         }
      ],
      "types" : [ "establishment" ],
      "url" : "https://plus.google.com/109147451404896083644/about?hl=es-ES",
"utc_offset" : 120,
      "vicinity" : "Calle de la Paz, 11, Madrid",
"website" : "http://teatroalbeniz.blogspot.com/"
},
   "status" : "OK"
}
```


2.6.2 API de rutas de Google

El API de rutas de Google es un servicio que permite calcular la ruta para llegar de una ubicación a otra mediante una solicitud HTTP [21].

La ubicación de origen y destino de la ruta se pueden expresar con el nombre de un pueblo o ciudad, con una dirección determinada o como coordenadas de latitud-longitud. Además es posible añadir paradas intermedias entre las dos ubicaciones.

Este servicio está diseñado para calcular rutas a partir de direcciones estáticas para la ubicación del contenido de la aplicación en un mapa. Sin embargo, este servicio no está diseñado para responder en tiempo real a la información introducida por el usuario.

La solicitud al API de rutas tiene el siguiente formato:

http://maps.googleapis.com/maps/api/directions/*output*?*parameters*

La variable output indica el formato en el que se va a devolver el resultado de la consulta. Puede ser uno de los dos valores que se indican a continuación:

- json: indica el formato de salida en Notación de objetos JavaScript.
- xml: indica el formato de salida como un archivo XML.

Como en la solicitud se pueden incluir datos sensibles para el usuario, como es su ubicación, también se puede acceder al API de rutas a través de un protocolo HTTPS:

https://maps.googleapis.com/maps/api/directions/*output*?*parameters*

Parámetros de solicitud

Algunos parámetros son obligatorios y otros opcionales. Todos los parámetros se separan con el carácter '&'. Los parámetros admitidos y sus posibles valores.

Parámetros obligatorios

- origin: define la dirección o el valor de las coordenadas latitud-longitud de la ubicación desde la que quieras calcular las rutas.
- destination: define la dirección o el valor de las coordenadas latitud-longitud de la ubicación desde la que se quiera calcular las rutas.
- sensor: indica si la solicitud de indicaciones procede de un dispositivo con un sensor de ubicación. Este valor debe ser true o false.

Parámetros opcionales

- mode: En el cálculo de rutas podemos especificar el tipo de transporte que vamos a utilizar para el desplazamiento. Este parámetro es el encargado de hacerlo. Los valores que puede tomar son:
    - driving (predeterminado) proporciona rutas estándar para llegar en coche a través de la red de carreteras.
    - walking solicita rutas a pie a través de aceras y rutas peatonales.
    - bicycling solicita rutas para llegar en bicicleta a través de carriles bici y vías preferenciales para bicicletas.
    - transit solicita indicaciones a través de rutas de transporte público (según disponibilidad). Si se escoge este modo de transporte también se debe especificar una hora de salida (departure_time) o una hora de llegada (arrival_time).
- Es posible que las rutas peatonales y para ciclistas no sean claras. En esos casos, aparecerán *warnings* en los resultados devueltos para las rutas sobre cómo llegar a pie o en bicicleta.
- waypoints: Esta opción indica un número de ubicaciones específicas por las que debe pasar la ruta. Para especificar un hito en la ruta se puede hacer mediante coordenadas de latitud-longitud o como una dirección que se codificará de forma geográfica. Los hitos solo son válidos para las indicaciones en coche, a pie y en bicicleta.
    - Se pueden poner varios hitos en la ruta, hasta un total de ocho, cada hito estará separado por el carácter '|'. Por defecto se calculará la ruta desde el origen a cada uno de los hitos hasta llegar al destino respetando el orden en el que se han introducido estos en la variable. Opcionalmente, se puede introducir optimize:true como el primer argumento del parámetro waypoints, de manera que el servicio de rutas reordene los hitos de forma más eficaz para optimizar la ruta proporcionada.

- alternatives: si se establece en true, indica que el servicio de rutas puede devolver más de una ruta alternativa.
- avoid: indica que la ruta o las rutas calculadas deben evitar determinados elementos. A continuación, se indican los dos argumentos que admite actualmente este parámetro.
    - tolls indica que la ruta calculada debe evitar los peajes de carretera y de puentes.
    - highways indica que la ruta calculada debe evitar las autopistas y las autovías.
- units: especifica el sistema de unidades que se utilizará para mostrar los resultados. Los posibles valores son:
    - metric indica el uso del sistema métrico. Las distancias en formato textual se devuelven en kilómetros y en metros.
    - imperial indica el uso del sistema imperial (británico). Las distancias en formato textual se devuelven en millas y en pies.
- region: es el código de país, especificado como un valor de dos caracteres ccTLD ("dominio de nivel superior").
- departure_time especifica la hora de salida que se quiere para las indicaciones en trasporte público. Se mide en segundos a partir de la medianoche del 1 de enero de 1970 UTC. Es el mismo formato que se utiliza en los ordenadores cuando se toma la fecha en formato numérico.
- arrival_time especifica la hora de llegada que se quiere para las indicaciones en trasporte público. Se mide en segundos a partir de la medianoche del 1 de enero de 1970 UTC. Es el mismo formato que se utiliza en los ordenadores cuando se toma la fecha en formato numérico.
- Los parámetros arrival_time y departure_time solo se aplican a las indicaciones en tránsito, y en ese supuesto tienen carácter obligatorio.

Elementos de las respuestas de rutas

Las respuestas de rutas constan de los siguientes elementos raíces:

- status: Contiene el estado de la solicitud y puede incluir información sobre depuración para ayudarte a descubrir el motivo por el que no funciona el servicio de rutas. El campo status puede contener los siguientes valores:
    - OK: indica que la respuesta contiene un resultado (result) válido.
    - NOT_FOUND: indica que al menos una de las ubicaciones especificadas en el origen, el destino o los hitos de la solicitud no se pudo codificar de forma geográfica.
    - ZERO_RESULTS: indica que no se pudo encontrar ninguna ruta entre el origen y el destino.

- MAX_WAYPOINTS_EXCEEDED: indica que se proporcionaron demasiados hitos (waypoints) en la solicitud. El número máximo permitido para waypoints es 8, además del origen y del destino.
- INVALID_REQUEST: indica que la solicitud enviada no era válida.
- OVER_QUERY_LIMIT: indica que el servicio ha recibido demasiadas solicitudes de la aplicación en el tiempo permitido. Para el uso público del API se pueden hacer hasta un total de 2.500 peticiones de ruta al día desde una misma aplicación.
- REQUEST_DENIED: indica que el servicio ha denegado el uso a la aplicación.
- UNKNOWN_ERROR: indica que no se ha podido procesar una solicitud de rutas debido a un error del servidor.

- routes: Cuando el API de rutas devuelve resultados, los ubica en un conjunto de routes. Aunque el servicio no devuelva ningún resultado (por ejemplo, si no existe la dirección), el API devolverá un conjunto de routes vacío.

  - Cada elemento del conjunto de routes contiene un resultado único del origen y del destino especificados. Esta ruta puede constar de uno o varios legs, en función de los hitos que se hayan incluido en la petición. Además, la ruta también incluye información de derechos de autor y advertencias que se deben mostrar al usuario junto con la información de la ruta.

  Cada ruta del campo routes puede contener los siguientes campos:

  - Summary: contiene una breve descripción textual sobre la ruta que permita identificarla y distinguirla de otras alternativas.
  - legs[]: contiene un conjunto que consta de información sobre un tramo de la ruta comprendido entre dos hitos de la ruta proporcionada. Una ruta sin hitos contendrá un único tramo dentro del conjunto de legs. Cada tramo consta de una serie de pasos (steps).
  - waypoint_order: incluye un conjunto que indica el orden de los hitos de la ruta calculada. Estos hitos se pueden volver a ordenar si en la solicitud se transmitió optimize:true: en el parámetro waypoints.
  - overview_polyline: contiene un objeto que consta de un conjunto de puntos (points) codificados que representan una ruta aproximada (suavizada) de las indicaciones resultantes.
  - Bounds: contiene el cuadro delimitador de la ventana gráfica de esta ruta.
  - Copyrights: contiene el texto de los derechos de autor que se mostrará con la ruta.
  - warnings[]: contiene un conjunto de advertencias que se visualizará cuando se muestren las rutas.

Cada elemento del conjunto legs especifica un tramo único del trayecto desde el origen al destino de la ruta calculada. Las rutas que no contengan hitos constarán de un único

"tramo", mientras que las rutas en las que se hayan definido uno o varios hitos constarán de uno o varios tramos correspondientes a los tramos específicos del trayecto.

Cada tramo del campo legs puede contener los siguientes campos:

- steps[]: Define un paso único de las rutas calculadas. Un paso es la unidad más atómica de una ruta, que consta de un único paso que describe una instrucción específica y única del trayecto. Contiene información sobre la distancia y sobre el tiempo con respecto al paso siguiente.

  Cada paso del campo steps puede contener los siguientes campos:

  - html_instructions: contiene instrucciones para el usuario, presentadas en forma de cadena de texto HTML.
  - Distance: contiene la distancia que hay que recorrer desde un paso hasta el siguiente.
  - Duration: contiene el tiempo normal necesario para realizar un paso antes de pasar al siguiente. Es posible que no se conozca la duración y que este campo no esté definido.
  - start_location: es un conjunto único de campos de lat y de lng que indica la ubicación del punto de partida de un paso determinado.
  - end_location: es un conjunto único de campos de lat y de lng que indica la ubicación del punto de partida de un paso determinado.
  - sub_steps: contiene indicaciones detalladas para ir a pie o indicaciones para ir en transporte público. Los subpasos solo están disponibles cuando travel_mode: se establece en transit. El conjunto sub_steps es del mismo tipo que steps.
  - transit_details: contiene información específica del transporte público. Este campo solo se muestra cuando travel_mode se establece en transit.

Las rutas de transporte público devuelven información adicional que no es relevante para otros medios de transporte. Estas propiedades adicionales se exponen a través del objeto transit_details, que, como hemos visto, es un elemento del conjunto steps[]. A partir del objeto TransitDetails, se puede acceder a información adicional sobre la parada, la línea y la compañía de transporte público.

Un objeto transit_details puede contener los campos indicados a continuación.

- arrival_stop y departure_stop: contienen información sobre la parada o la estación para esta parte del recorrido. Los detalles de la parada pueden incluir:
  - name: nombre de la estación o la parada
  - location: ubicación de la estación o parada de transporte público, representada como campos lat y lng.
- arrival_time y departure_time: contienen las horas de salida y de llegada para esta parte del viaje, con las siguientes tres propiedades:

    - text: valor de tiempo especificado como cadena. El valor de tiempo se muestra en la zona correspondiente situada en la parte superior de la parada de transporte público.
    - value: hora especificada con el formato Unix o los segundos desde la medianoche del 1 de enero de 1970 UTC.
    - time_zone: contiene la zona de uso horario en la que se encuentra esta estación.
- headsign: especifica la dirección en la que viaja la línea, tal y como aparece en el vehículo o en la parada de salida. Generalmente esta será la última estación.
- headway: especifica el número previsto de segundos entre salidas de la misma parada en ese momento.
- num_stops: contiene el número de paradas de este paso, contando la parada de llegada, pero no la de salida.
- Line: contiene información sobre la línea de transporte público utilizada en este paso y puede incluir las siguientes propiedades:
    - name: contiene el nombre completo de la línea de transporte público.
    - short_name: contiene el nombre abreviado de la línea de transporte público.
    - color: contiene el color que se utiliza normalmente para señalizar la línea de transporte público en cuestión. El color se especificará como una cadena hexadecimal con el formato: #RRGGBB.
    - agencies contiene un conjunto de objetos TransitAgency, cada uno de los cuales proporciona información sobre el operador de la línea, incluidas las siguientes propiedades:
        - name contiene el nombre de la empresa de transporte público.
        - url contiene la URL de la empresa de transporte público.
        - phone contiene el número de teléfono de la empresa de transporte público.
    - url contiene la URL de esta línea de transporte público tal y como la proporciona la empresa de transporte.
    - icon contiene la URL correspondiente al icono asociado a esta línea.
    - text_color contiene el color del texto que se utiliza normalmente para señalizar la línea en cuestión. El color se especificará como una cadena hexadecimal.
    - vehicle contiene el tipo de vehículo utilizado en la línea en cuestión. Puede incluir las siguientes propiedades:
        - name contiene el nombre del vehículo de la línea.
        - type contiene el tipo de vehículo utilizado en la línea en cuestión.
        - icon contiene la URL correspondiente a un icono asociado al tipo de vehículo en cuestión.

Ejemplo de petición para ir andando desde la Puerta del Sol hasta la calle Mayor en Madrid y que la respuesta sea en formato JSON:

http://maps.googleapis.com/maps/api/directions/json?origin=Puerta+sol,madrid&destination=plaza+mayor,madrid&region=es&sensor=false&mode=walking

Respuesta a la petición:


```
{
   "routes" : [
      {
         "bounds" : {
            "northeast" : {
               "lat" : 40.4168712,
               "lng" : -3.7033701
            },
            "southwest" : {
               "lat" : 40.4153381,
               "lng" : -3.707396
            }
         },
         "copyrights" : "Datos de mapa ©2013 Google, basado en BCN IGN España",
         "legs" : [
            {
               "distance" : {
                  "text" : "0,5 km",
                  "value" : 456
               },
               "duration" : {
                  "text" : "6 min",
                  "value" : 333
               },
               "end_address" : "Plaza Mayor, Madrid, España",
               "end_location" : {
                  "lat" : 40.4153381,
                  "lng" : -3.707396
               },
               "start_address" : "Puerta del Sol, 28013 Madrid, España",
               "start_location" : {
                  "lat" : 40.4168712,
                  "lng" : -3.7033981
               },
               "steps" : [
                  {
                     "distance" : {
                        "text" : "14 m",
                        "value" : 14
                     },
                     "duration" : {
                        "text" : "1 min",
                        "value" : 9
                     },
                     "end_location" : {
                        "lat" : 40.4167522,
                        "lng" : -3.7033701
                     },
                     "html_instructions" : "Dirígete al \u003cb\u003esur\u003c/b\u003e por
\u003cb\u003eCalle Montera\u003c/b\u003e hacia \u003cb\u003ePlaza Puerta del
Sol\u003c/b\u003e",
                     "polyline" : {
                        "points" : "m|tuFfirUFAF?FC"
                     },
                     "start_location" : {
                        "lat" : 40.4168712,
                        "lng" : -3.7033981
                     },
                     "travel_mode" : "WALKING"
                  },
                  {
                     "distance" : {
                        "text" : "94 m",
                        "value" : 94
                     },
```

```
                     "duration" : {
                        "text" : "1 min",
                        "value" : 63
                     },
                     "end_location" : {
                        "lat" : 40.416626,
                        "lng" : -3.7044618
                     },
                     "html_instructions" : "Gira a la \u003cb\u003ederecha\u003c/b\u003e
hacia \u003cb\u003ePlaza Puerta del Sol\u003c/b\u003e",
                     "maneuver" : "turn-right",
                     "polyline" : {
                        "points" : "u{tuF`irU?VF~@@ZFz@Dh@"
                     },
                     "start_location" : {
                        "lat" : 40.4167522,
                        "lng" : -3.7033701
                     },
                     "travel_mode" : "WALKING"
                  },

                  (…)

               ],
               "via_waypoint" : []
            }
         ],
         "overview_polyline" : {
            "points" : "m|tuFfirUNAFC?VHzALdBhArNv@MN?~@j@^CLA@~@"
         },
         "summary" : "Calle Mayor",
         "warnings" : [
            "Las rutas a pie están en versión beta. Ten cuidado. -  En esta ruta puede que
no haya aceras o pasos para peatones."
         ],
         "waypoint_order" : []
      }
   ],
   "status" : "OK"
}
```


2.6.3 API de Kulturkilk, Página web de eventos culturales País Vasco

Un ejemplo de página web en las que se encuentran eventos culturales y que implementa un API para recibir peticiones es http://www.kulturklik.euskadi.net/. Las consultas a la página se hacen mediante peticiones HTTP y las respuestas tendrán un formato JSON [4].

Los parámetros para implementar la consulta de eventos son:

- api_call: Este parámetro es obligatorio y siempre toma el valor events. Es indicatorio de que se está realizando una petición.
- from: Indica la fecha a partir de la cual se obtendrán eventos. El formato de fecha que utiliza es yyyy-mm-dd. Si se omite este parámetro se tomará como fecha inicial la actual.
- to: Indica la fecha hasta la que se obtendrán eventos. El formato de fecha es el mismo que en el caso anterior, yyyy-mm-dd. Si no se incluye se considerará como fecha fin la actual.

- lang: Establece el Idioma en el que se desea que se devuelvan los datos del evento. Admite los valores eu para euskera y es para castellano.
- max: Indica el número máximo de eventos que se quieren obtener hasta un total de 2000. Si no se especifica se tomará como máximo 2000.
- minx: Indica el mínimo valor de longitud de las coordenadas de los municipios en los que buscar eventos. Si no se incluye se devolverán eventos de cualquier municipio.
- maxx: Máximo valor de longitud de las coordenadas de los municipios en los que buscar eventos.
- miny: Mínimo valor de latitud de las coordenadas de los municipios en los que buscar eventos.
- maxy: Máximo valor de latitud de las coordenadas de los municipios en los que buscar eventos.

Nótese que en este caso se puede determinar áreas geográficas en forma de rectángulo, ya que las coordenadas se dan como valores máximos y mínimos de la longitud y la latitud. Esto es diferente a lo que sucede en googlemaps, por ejemplo, que se dan áreas geográficas con forma de círculo al establecerse un punto y un radio.

La respuesta, como se ha comentado antes, va a tener un formato JSON y va a consistir en un *array* con dos posiciones:

- count: Indica el número de eventos devueltos
- eventos: Devuelve un *array* de eventos, cada elemento contiene campos con información sobre el evento:
    - evento_titulo: Título del evento
    - evento_url: URL del evento dentro de la página Kulturklik
    - evento_tipo: Tipo del evento. Los posibles valores en castellano son: Concierto, Teatro, Exposición, Danza, Musical, Ópera, Presentación, Bertsolarismo, Conferencia, Recital, Cuenta cuento, Feria de artesanía, Feria del libro, Títeres, Proyección audiovisual, Jornadas, Curso-taller, Concurso, Festival, Actividad infantil, Magia, Circo, Pastoral, Payasos, Monólogos, Humor.
    - longitude: Coordenada longitud del municipio donde se celebra el evento.
    - latitude: Coordenada latitud del municipio donde se celebra el evento.

En la petición no se puede especificar un tipo de evento concreto que buscar. Hay que examinar la respuesta a posteriori.

Algunos ejemplos de peticiones y respuestas:

- Petición sin condiciones:

http://www.kulturklik.euskadi.net/?api_call=events&lang=es

Nos devolverá todos los eventos previstos para hoy en todo el País Vasco. A continuación podemos ver un fragmento. Si analizamos el primer resultado vemos que tenemos una exposición con el título "Cazadores de ballenas" en la localización -1.98º Este, 43,32º Norte y la información del evento la podemos encontrar en la página web:

http://www.kulturklik.euskadi.net\/?lang=es&p=80526&utm_source=agenda&utm_medium=api&utm_campaign=suscripciones-agenda


```
{"count":207,"eventos":[{"evento_titulo":"&quot;Cazadores de
ballenas&quot;","evento_url":"http:\/\/www.kulturklik.euskadi.net\/?lang=es&p
=80526&utm_source=agenda&utm_medium=api&utm_campaign=suscripciones-
agenda","evento_tipo":"Exposici\u00f3n","longitude":"-
1.984449","latitude":"43.320725"},{"evento_titulo":"&quot;Gipuzkoa en
miniatura&quot;","evento_url":"http:\/\/www.kulturklik.euskadi.net\/?lang=es&
p=137679&utm_source=agenda&utm_medium=api&utm_campaign=suscripciones-
agenda","evento_tipo":"Exposici\u00f3n","longitude":"-
1.984449","latitude":"43.320725"}, (…)]}
```


- En este otro ejemplo vamos a buscar los eventos que vayan a producirse entre las fechas 28/09/2012 y 30/09/2012, que devuelva un máximo de 3 resultados y que se produzcan entre unas coordenadas específicas:

http://www.kulturklik.euskadi.net/?api_call=events&from=2012-09-28&to=2012-09-30&lang=es&max=3&maxy=43.4&miny=43.3&maxx=-1.9&minx=-2

La respuesta que nos devuelve la petición es:


```
{"count":3,"eventos":[{"evento_titulo":"&quot;Cazadores de
ballenas&quot;","evento_url":"http:\/\/www.kulturklik.euskadi.net\/?lang=es&p
=80526&utm_source=agenda&utm_medium=api&utm_campaign=suscripciones-
agenda","evento_tipo":"Exposici\u00f3n","longitude":"-
1.984449","latitude":"43.320725"},{"evento_titulo":"&quot;Gipuzkoa en
miniatura&quot;","evento_url":"http:\/\/www.kulturklik.euskadi.net\/?lang=es&
p=137679&utm_source=agenda&utm_medium=api&utm_campaign=suscripciones-
agenda","evento_tipo":"Exposici\u00f3n","longitude":"-
1.984449","latitude":"43.320725"},{"evento_titulo":"Pantalla Global:
&quot;Cuando la pantalla se convierte en mundo \/ Cuando el mundo se
convierte en
pantalla&quot;","evento_url":"http:\/\/www.kulturklik.euskadi.net\/?lang=es&p
=230116&utm_source=agenda&utm_medium=api&utm_campaign=suscripciones-
agenda","evento_tipo":"Exposici\u00f3n","longitude":"-
1.984449","latitude":"43.320725"}]}
```


2.6.4 Servicio web de búsqueda de cines iGoogle

Este servicio es parte de un *gadget* definido para iGoogle que actualmente se encuentra en desuso pero que sigue admitiendo consultas.

iGoogle fue una apuesta de Google para crear una página principal personalizada que incluía un cuadro de búsqueda en la parte superior y todos los *gadgets* que se quisiera en la parte inferior. El 3 de julio de 2012 se confirmó la desaparición de iGoogle, el 31 de julio de 2012 desapareció su versión móvil y a partir del 1 de noviembre de 2013 se eliminará totalmente. Google declaró que esta decisión se tomó por la innecesidad del

servicio que se ha dado paulatinamente desde la aparición de aplicaciones con plataformas modernas como Android [22].

Actualmente el *gadget* que utilizaba esta consulta ya no funciona desde igoogle.

La consulta que se puede realizar a este servicio web acepta varios parámetros diferentes [23]:

- movies: Este parámetro es obligatorio. Establece la ciudad en la cual buscar los cines.

- theater: Establece el nombre o parte del nombre de un cine.

- date: Indica el día para el cuál queremos ver los horarios. Su valor es un entero: '0' para hoy, '1' para mañana, '2' para pasado mañana, etc.

- time: Indica la sesión para la que se quiere los horarios. Su valor es un entero: '1' para la sesión matinal, '2' para la sesión de tarde, '3' para la sesión de noche, '4' para la madrugada y '5' para cualquiera

- start: Indica el número de lista por el que se empiezan a mostrar los resultados.

La implementación de esta consulta es muy básica y sólo permite mostrar tres elementos de la lista completa que genera. Podemos utilizar el parámetro start para ir avanzando en la lista poco a poco.

La respuesta nos ofrece las películas que hay en cartelera en cada cine y cierta información sobre ellas. Algunos de los campos son:

- title: Título de la película
- length: Duración de la película. Se da en formato String indicando el número de horas y minutos
- genre: Género de la película
- num_reviews: Número de críticas almacenadas sobre esta película
- reviews_url: URL en dónde se pueden leer las críticas de la película
- landingpage_url: Dirección que se ha de añadir a la url www.google.es para ver la información de la película en la página de google
- reviews_rating_percent: Valoración media que le han dado los usuarios que han escrito una crítica

Un ejemplo de consulta sería la siguiente:

http://www.google.com/ig/api?movies=madrid&theatre=ideal&date=0&time=3&start=0

En la que estaríamos buscando las películas que se emiten en el cine Ideal de Madrid hoy en la sesión nocturna. La respuesta sería:

```
<xml_api_reply version="1">
   <movies module_id="0" tab_id="0" mobile_row="0" mobile_zipped="1" row="0" section="0">
         <location data="madrid"/>
         <showtimes_url data="/movies?client=ig&near=madrid"/>
      <movie>
         <title data="Las aventuras de Tadeo Jones"/>
         <movie_id data="8733232946361385343"/>
         <length data="1h 25min"/>
         <genre data="Animación"/>
         <mpaarating data=""/>
         <num_reviews data="1 críticas"/>
         <reviews_url data="/movies?hl=es&near=madrid&mid=7932ad18149bc17f#reviews"/>
         <imdb_url data=""/>
         <landingpage_url data="/movies?hl=es&near=madrid&mid=7932ad18149bc17f"/>
         <reviews_rating_percent data="0"/>
      </movie>
      <movie>
         <title data="Total Recall (Desafío total)"/>
         <movie_id data="-8084784525435178387"/>
         <length data="2h 1min"/>
         <genre data="Acción"/>
         <mpaarating data=""/>
         <num_reviews data="2 críticas"/>
         <reviews_url data="/movies?hl=es&near=madrid&mid=8fcd134f23dc726d#reviews"/>
         <imdb_url data=""/>
         <landingpage_url data="/movies?hl=es&near=madrid&mid=8fcd134f23dc726d"/>
         <reviews_rating_percent data="60"/>
      </movie>
      <movie>
         <title data="The Possession (El Origen del Mal)"/>
         <movie_id data="2801916491756555496"/>
         <length data="1h 36min"/>
         <genre data="Terror"/>
         <mpaarating data=""/>
         <num_reviews data="0 críticas"/>
         <reviews_url data=""/>
         <imdb_url data=""/>
         <landingpage_url data="/movies?hl=es&near=madrid&mid=26e268343d754ce8"/>
         <reviews_rating_percent data=""/>
      </movie>
   </movies>
</xml_api_reply>
```

Este servicio web no dispone de un API pública específica y tiene unas prestaciones muy limitadas.

2.7 Página web de carteleras de cines de Google

No existe un API que ofrezca información sobre carteleras de cine en España en función de la ubicación del usuario. Por otro lado sí que existen páginas web que contienen este tipo de datos ligados a una posición concreta. Dependiendo del formato de una web, de su nivel de complejidad y de la forma en la que se hacen las llamadas dentro de ella, es factible parsear dicha página web para extraer la información deseada incorporada en ella [5].

Google dispone de una página sencilla y bien estructurada sobre cines y cartelera en función de una localización.

2.7.1 Parseo de páginas web en Android (JSOUP)

Jsoup es una librería de Java que funciona sobre páginas HTML independientemente de si están correctamente formadas o no. Contiene un API para extraer y manipular la información proveniente de la página web y convertirlo en un árbol de objetos [24].

Las clases principales en las que se va a transformar la estructura de la página web son Node, TextNode, Element y Document.

Document hereda de la clase Element. Al parsear una página web obtendremos un objeto Document que contendrá uno o varios objetos Element. Element a su vez hereda del interfaz Node y representa a cada una de las etiquetas de un HTML con sus atributos, sus valores y sus nodos hijos. TextNode también hereda de Node y representa a las etiquetas que contienen texto. También existe una clase Elements que consiste en una lista de la clase Element.

Se puede realizar el parseo de HTML a partir de un String con la página HTML completa o de sólo una parte de ésta, de un fichero o de una URL utilizando respectivamente los métodos estáticos:

- Jsoup.parse(String html);
- Jsoup.parseBodyFragment(String html);
- Jsoup.connect(String url).get();
- Jsoup.parse(File in, String charsetName);

Todos estos métodos devuelven un objeto Document que describe el árbol de información del HTML.

Para obtener los datos almacenados en este objeto Document existen varios métodos de tipo DOM (*Document Object Model*), que permiten encontrar elementos, extraerlos y manipular sus datos. Podemos buscar un elemento por su identificador, clase, etiqueta, atributo, etc. o podemos movernos por el árbol creado buscando elementos padre, hijos, hermanos, etc. del elemento actual seleccionado. Una vez seleccionado un elemento podemos extraer de él o manipular toda la información que contiene.

Además de los métodos de tipo DOM, también se pueden utilizar otros basados en CSS o del tipo *jquery* para encontrar los elementos que se buscan. Para este tipo de búsquedas se utiliza el método estático select(String selector) que se puede ejecutar sobre la clase Element o sobre Elements. El String selector especifica la búsqueda. Los diferentes formatos básicos que puede seguir el selector son:

- tagname: encuentra elementos por la etiqueta, ejemplo: a

- ns|tag: encuentra elementos por la etiqueta dentro de un determinado namespace, ejemplo: fb|name encuentra elementos del tipo <fb:name>
- #id: encuentra elementos por su ID, ejemplo: #logo
- .class: encuentra elementos en function del tipo de clase, ejemplo:.masthead
- [attribute]: devuelve elementos con atributos, ejemplo: [href]
- [^attr]: devuelve elementos que contengan un atributo con el prefijo, ejemplo: [^data-] encuentra elementos con atributos de HTML5
- [attr=value]: devuelve elementos cuyo atributo tenga ese valor, ejemplo:[width=500]
- [attr^=value], [attr$=value], [attr*=value]: devuelve elementos que tengan atributos que empiecen, terminen o contengan el valor, ejemplo: [href*=/path/]
- [attr~=regex]: devuelve elementos con atributos cuyo valor contenga la expresión; ejemplo: img[src~=(?i)\.(png|jpe?g)]
- * : devuelve todos los elementos

Todos estos formatos se pueden combinar entre ellos. Además de estos formatos de selección, existen otros que están basados en la estructura del árbol que se crea:

- :lt(n): encuentra elementos hermanos cuya distancia relativa en el árbol sea menor de n, ejemplo: td:lt(3)
- :gt(n): encuentra elementos hermanos cuya distancia relativa en el árbol sea mayor de n; ejemplo: div p:gt(2)
- :eq(n): encuentra elementos hermanos cuya distancia relativa en el árbol sea igual a n; ejemplo: form input:eq(1)
- :has(seletor): encuentra elementos que contengan elementos que concuerden con la selección; ejemplo: div:has(p)
- :not(selector): encuentra elementos que no contengan elementos que concuerden con la selección; ejemplo: div:not(.logo)
- :contains(text): encuentra elementos que contengan el texto dado. La búsqueda distingue entre mayúsculas y minúsculas; ejemplo: p:contains(jsoup)
- :containsOwn(text): encuentra elementos cuyo texto sea igual al dado
- :matches(regex): encuentra elementos que contengan textos que concuerden con una expresión dada; ejemplo: div:matches((?i)login)
- :matchesOwn(regex): encuentra elementos cuyo propio texto concuerde con la expression dada.

Un ejemplo de uso de *jsoup* podría ser el obtener todas las imágenes de una determinada página web:

```
public class ListLinks {
   public static void main(String[] args) throws IOException {
     Validate.isTrue(args.length == 1, "usage: supply url to fetch");
     String url = args[0];
```

```
     print("Fetching %s...", url);

     Document doc = Jsoup.connect(url).get();
     Elements media = doc.select("[src]");

     for (Element src : media) {
       if (src.tagName().equals("img")){
         print(" * %s: <%s> %sx%s (%s)",
            src.tagName(), src.attr("abs:src"), src.attr("width"), src.attr("height"),
                      trim(src.attr("alt"), 20));
       }
     }

   }

    private static void print(String msg, Object... args) {
        System.out.println(String.format(msg, args));
    }

    private static String trim(String s, int width) {
        if (s.length() > width)
            return s.substring(0, width-1) + ".";
        else
            return s;
    }
}
```

Introduciendo como *url* http://news.ycombinator.com/, obtenemos la siguiente salida:

```
 * img: <http://ycombinator.com/images/y18.gif> 18x18 ()
 * img: <http://ycombinator.com/images/s.gif> 10x1 ()
 * img: <http://ycombinator.com/images/grayarrow.gif> x ()
 * img: <http://ycombinator.com/images/s.gif> 0x10 ()
 * img: <http://ycombinator.com/images/s.gif> 15x1 ()
 * img: <http://ycombinator.com/images/hnsearch.png> x ()
 * img: <http://ycombinator.com/images/s.gif> 25x1 ()
 * img: <http://mixpanel.com/site_media/images/mixpanel_partner_logo_borderless.gif> x
(Analytics by Mixpan.)
```

2.7.2 Página web de www.google.es/movies

Google dispone de una página web específica sobre cines y cartelera: www.google.com/movies.

Esta página web proporciona la información de la cartelera de una gran cantidad de cines, ofreciendo qué películas se están proyectando actualmente y los horarios de los pases de éstas. El motor de búsqueda de la cartelera utiliza la proximidad a un lugar determinado como criterio para mostrar la información de los cines más cercanos.

La página web tiene el aspecto de la figura 2.7.

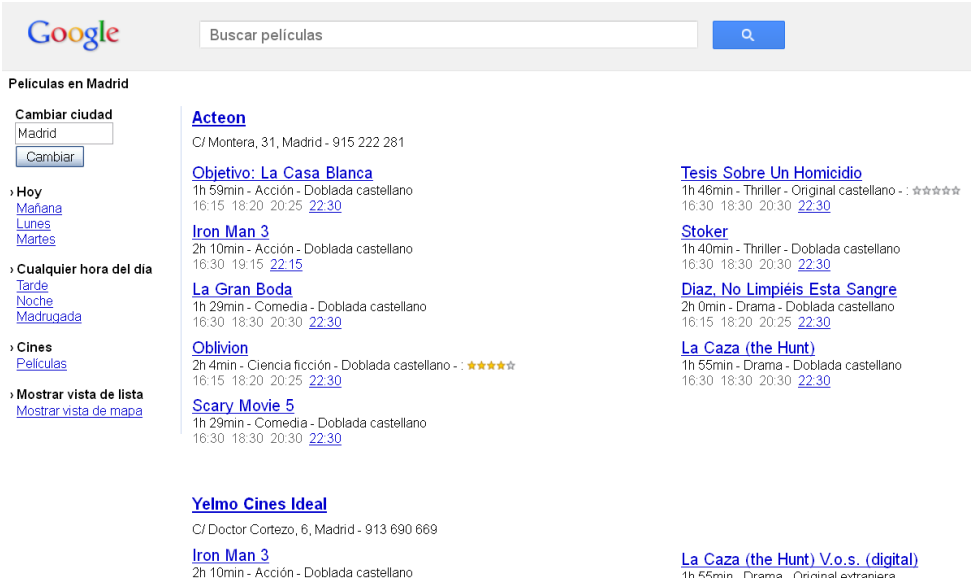

Figura 2.7 Apariencia de la página web de carteleras de Google

De inicio y por defecto, la página obtiene la información de localización en la red a través de nuestra dirección MAC, con ello ubica el lugar de conexión y muestra los resultados de la cartelera en función de éste.

El resultado de la consulta presenta una serie de cines ordenados por cercanía. Cada uno de los cines está encabezado por su nombre y por información relativa a éste, como puede ser su localización, teléfono, etc. Debajo de este encabezado aparecen dos columnas con las películas que se están proyectando actualmente en dicho cine. Cada película viene subtitulada con características, como puede ser la duración, su género o si es versión original o doblada. Debajo del título y de sus propiedades aparecen los horarios de los pases. Los horarios son enlaces que en la mayoría de los casos permiten redirigir a entradas.com para comprar la entrada a la sesión seleccionada.

La información que se proporciona tanto del cine como de las películas depende de los datos que cede cada cine a Google y por lo tanto, aunque sigue un modelo determinado, no siempre aparecen todas las características. Por la misma razón no aparecen todos los cines, aunque sí la gran mayoría.

La página incluye varias opciones para definir la búsqueda:

- Localización: Incluye un campo de entrada de texto con el título “Cambiar ciudad” en el que se puede introducir el lugar geográfico en torno al cual se va a realizar la búsqueda. En este campo se puede introducir, aparte del nombre de una ciudad, una dirección concreta o una coordenada geográfica.

- Campo de búsqueda: Permite realizar búsquedas de cines por nombre y de películas por título. En el primer caso el resultado se muestra en pantalla con el mismo formato que en el caso general, con la diferencia de que se refleja un sólo cine. En el segundo caso aparece el título de la película como cabecera, subtitulado con sus características, y debajo se listan los cines en los que se proyecta esta película siguiendo el criterio de proximidad.

- Elección de fecha: Se puede elegir la fecha en la que queremos ver los horarios de las películas. Por defecto está seleccionado el día actual marcado como "Hoy" y se puede consultar la cartelera para otros tres días en adelante.

- Elección de sesión: Filtra la cartelera por tipo de sesión, por lo que aparecen únicamente los resultados para el momento que nos interese. Por defecto aparecen todos los horarios, pero se puede filtrar por sesión de mañana, tarde, noche o madrugada.

- Vista general por cines o películas: Siguiendo el único criterio de la proximidad, podemos optar por ver la información de la cartelera ordenada por cines, en los que se listarán las películas que se proyectan en cada uno, o ver la información ordenada por películas, en las que se listarán los cines en los que se proyecta.

- Mostrar vista de lista o de mapa: Si se elige la opción de ver la cartelera ordenada por cines, se puede elegir entre mostrar los resultados en forma de lista, como se ha visto antes, o en forma de lista reducida acompañada con un mapa con la situación de los cines.

- Elección de género: Si elegimos la opción de ver la cartelera ordenada por películas, se puede hacer un filtro por género. Por defecto aparecen todas, se puede escoger hacer el filtro por género de acción, animación, ciencia ficción, comedia, drama, terror, thriller u otros.

La página web de google.es/movies permite escoger las diferentes opciones de las que dispone añadiéndolas a la dirección http de ésta. La página además está bien construida y los resultados responden siempre a la misma estructura.

La llamada a la página web se hará con el siguiente formato:

http://www.google.es/movies?near=lat,lng&date=0&time=5

near: Indica la ubicación que se tomará como referencia para la búsqueda de los cines. Se puede indicar una dirección o unas coordenadas.

date: Indica la fecha en la que se quiere ver los horarios, el valor 0 correspondería a hoy, el 1 a mañana y así sucesivamente.

time: Indica el momento del día en el que se quiere ver la cartelera. El valor 1 corresponde a la mañana, el 2 al mediodía, 3 tarde, 4 noche y 5 al día completo.

2.7.3 Estructura de la página web de www.google.es/movies

Una vez formada la dirección http que nos devolverá la página con la información que se necesita, se ha de comprobar la estructura que tienen los datos en su código fuente para poder extraerlos con ayuda de la librería de JSOUP.

Los datos que interesan están mapeados en un árbol lógico cuyo nodo raíz es la etiqueta div de la clase movie_results que es un nodo hijo de la etiqueta results perteneciente al cuerpo del HTML. Los elementos importantes son:

- div class=movie_results: Es el elemento raíz del subárbol lógico que contiene toda la información sobre los cines que se quiere extraer.
- div class=theater: Este elemento es hijo del anterior, representa a un cine y tiene dos nodos hijos que almacenan la información del cine y las películas.
- div class=desc: Contiene la descripción del cine. Tiene dos nodos hijos:
- h2 class=name: Tiene como valor de texto el nombre del cine
- div class=info: Tiene como valor de texto información relevante al cine, como es su dirección, teléfono, etc.
- div class= showtimes: Contiene la información sobre las películas. Se divide en dos agrupaciones de películas, las que se muestran en la página a la derecha y las que se muestran a la izquierda. Estos nodos hijos vienen marcados como div class=show_left y div class=show_right.
- div class=show_* : Contiene varios elementos hijos del tipo div class=movie con la información de la película.
- div class=movie: Cada elemento contiene tres nodos hijos que almacenan el título de la película, información sobre ella y los horarios disponibles de sesión.
- div class=name: Tiene como valor de texto el título de la película.
- span class=info: Tiene como valor de texto información acerca de la película como la duración, el género o la versión.
- div class=times: Contiene la agrupación con los horarios de las sesiones disponibles. Sus nodos hijos son del tipo span style=color.
- span style=color: Representa a cada uno de los horarios. Tiene dos nodos hijos, el segundo que es del tipo a class=f1 contiene la información sobre el horario.
- a class=f1: Contiene como texto el horario de la sesión y contiene como hiperenlace la dirección de la página correspondiente de entradas.com en donde se puede comprar la entrada para esa sesión.

Capítulo 3. Descripción general del sistema

En este apartado se pretende proporcionar una visión global de la aplicación mediante una explicación en profundidad de la arquitectura y funcionalidad de la misma. Con este fin, se exponen los elementos que intervienen en el sistema y la forma que tienen de interactuar entre ellos para conseguir la funcionalidad requerida.

También se establece un esquema de los módulos en los que está dividido el proyecto introduciéndose la funcionalidad de cada uno de ellos y el esquema de sus clases.

Por último se especificarán los requisitos que se deben cumplir para utilizar la aplicación, los requisitos que se esperan de ella y los casos de uso.

3.1. Arquitectura

Como está representado en la figura 3.1, la aplicación interactúa con varios elementos para conseguir la funcionalidad requerida:

- Sistema de localización: El usuario a través de su terminal utilizará uno de los tres sistemas de localización que permite utilizar el dispositivo Android para determinar su ubicación (GPS, WPS, *Cell-ID*).
- Página web google.es/movies: El terminal se conecta a la página web de google.es/movies indicando la posición del usuario y extrae los datos, películas y horarios de los cines cercanos [5].
- Página web de entradas.com: Una vez que se ha seleccionado la sesión que se desea, se redirige al usuario a la página correspondiente dentro del dominio web entradas.com en el cual el usuario puede comprar directamente la entrada [25].

- Servicio web de Kulturklik: En el caso de encontrarse en el País Vasco, se hace uso del servicio web de Kulturklik para obtener las obras de teatro que se están representando cerca de éste [4].
- Página web de kultrklik.euskadi.net: A través de los datos obtenidos por el servicio web de Kulturklik se redirige al usuario a la página web correspondiente dentro del dominio de kulturklik.euskadi.net en el que se proporciona información adicional sobre la obra de teatro y redirige al usuario a la página web correspondiente en dónde se puede obtener la entrada [26].
- Servicio web de Google Directions: Este servicio proporciona al usuario la información de la ruta correspondiente a su posición y al lugar del evento escogido. Se puede configurar la petición para que se calcule la ruta en coche, andando o en transporte público [21].
- Servicio web de geolocalización: Este sistema se utiliza para transformar las coordenadas de un punto en una dirección y al contrario [9].

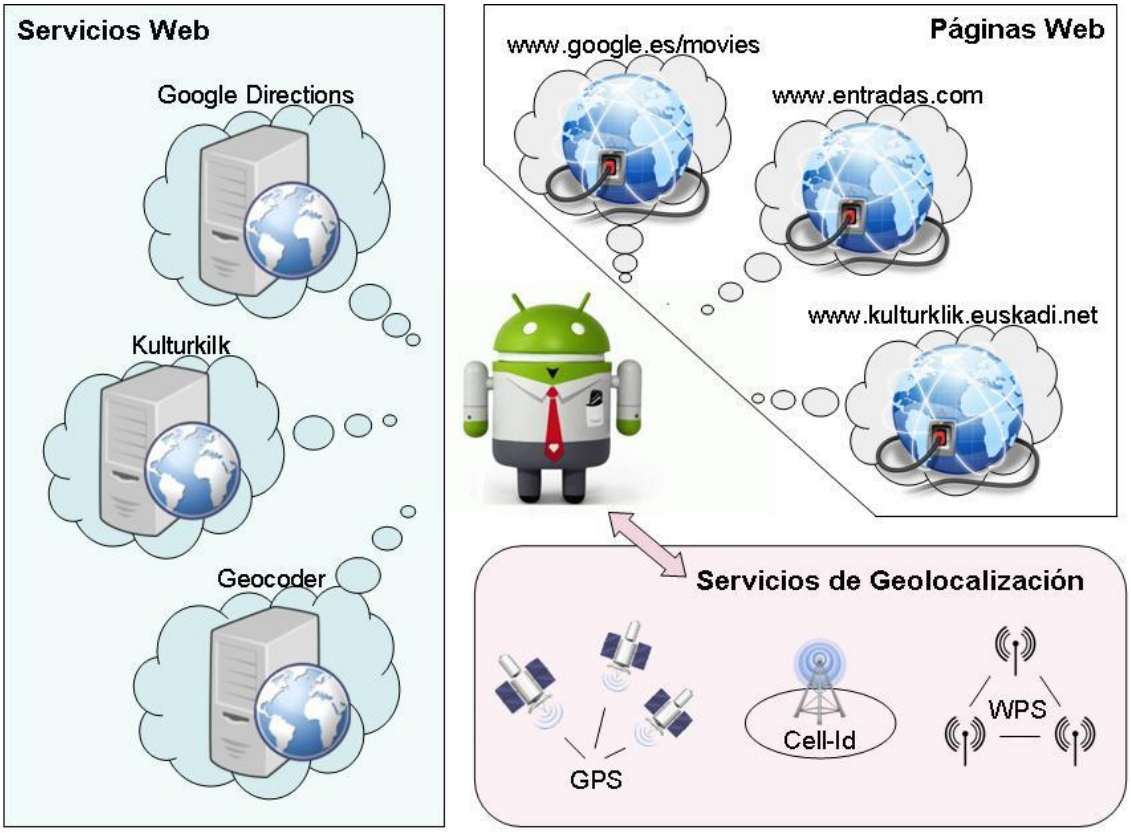

Figura 3.1 Usuario móvil y elementos de la arquitectura

3.2. Funcionalidad

3.2.1 Funcionalidad de la aplicación a desarrollar

El objetivo principal de la aplicación es proporcionar al usuario la posibilidad de comprar entradas para diferentes eventos basándose en su posición y preferencias.

Cuando el usuario accede a la aplicación se realiza una búsqueda sobre los distintos proveedores de información de eventos que se almacena en memoria. Una vez obtenidos los datos, el usuario tiene el control para elegir diferentes modos de visualizar la información.

Si se ha registrado un evento muy cerca de la posición de usuario, la aplicación mostrará directamente la información almacenada sobre ese evento. En el caso en el que se hayan encontrado varios eventos se muestra una lista seleccionable únicamente con esos eventos. Desde esta lista se puede seleccionar un evento en particular para acceder a su información, mostrar el menú principal o mostrar al usuario la posición de todos los eventos en un mapa. Este comportamiento se puede observar en el diagrama de flujo de la Figura 3.2.

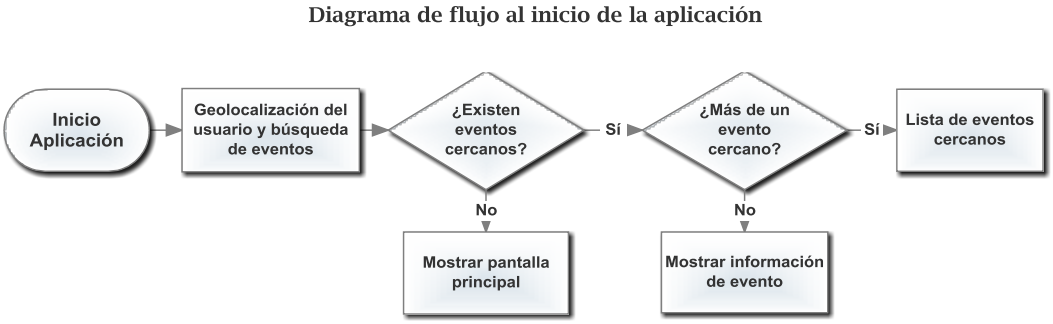

Figura 3.2 Diagrama de flujo de la rutina de inicio de la aplicación

La forma principal de mostrar la información al usuario es a través de un mapa. En este mapa se representa la posición del usuario junto a la posición de los cines y teatros cercanos a éste. El color del marcador será diferente dependiendo de si representa al usuario, a un cine o a un teatro.

Una vez colocados los marcadores, el usuario tiene la posibilidad de interactuar con ellos y acceder a los datos que la aplicación tiene guardada sobre los elementos a los que representan. Como se puede ver en la figura 3.4, la forma de exponer la información será distinta para cines que para teatros:

- En el caso de que el marcador escogido represente a un cine, se muestra una lista expandible con las películas en cartelera que contiene una lista con los horarios de las sesiones disponibles. La aplicación guarda las acciones realizadas por el usuario en la sesión actual y en las sesiones anteriores, de este modo se almacena

un perfil de usuario con las preferencias de éste y se ordena la lista de películas en función de sus elecciones. Al seleccionar uno de los horarios se redirige al usuario a la página web correspondiente para efectuar la compra de la entrada.

- En el caso de que el marcador escogido represente a un teatro, se redirigirá al usuario a la página web del evento a través de la cual se puede acceder a la página que permite la compra de la entrada.

Seleccionando un marcador sobre el mapa, el usuario también tiene la oportunidad de calcular la ruta desde el punto en el que se encuentra hasta la posición de dicho marcador. La ruta se representa sobreimpresa sobre el mapa y se puede acceder a la descripción escrita de la ruta paso a paso. La ruta se puede calcular para un desplazamiento en coche, andando o en transporte público. Para una misma posición de evento se mostrarán a la vez sobre el mapa los diferentes caminos que representan a los diferentes tipos de transporte de los que se haya pedido el cálculo de camino. Como se verá luego y se muestra en la figura 3.3, el cálculo de rutas puede ser invocado desde diferentes actividades.

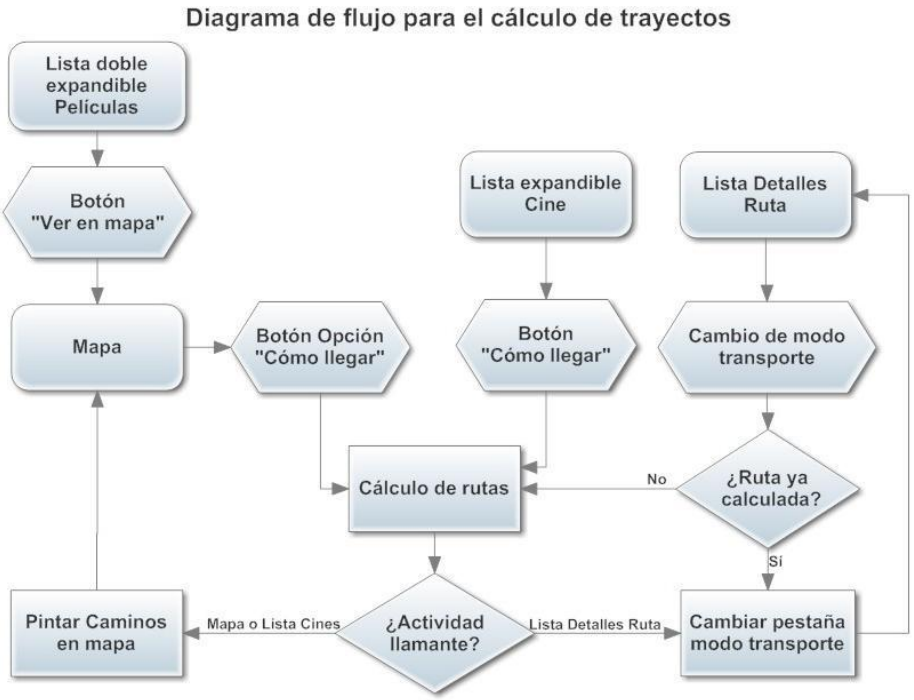

Figura 3.3 Diagrama de flujo para el cálculo de trayectos

Aparte de la representación sobre el mapa de las posiciones de los eventos, también está disponible una forma alternativa de mostrar la información para las sesiones de cine. El usuario puede ver la cartelera de los cines que están cerca en forma de listado de películas. Se listan las películas ordenadas por las preferencias del usuario de la misma manera que se hacían en el caso anterior.

Las películas se muestran en una doble lista expandible. Cada elemento correspondiente a una película contiene una lista con los nombres de los cines en los que se representa. Cada elemento que representa a uno de estos cines tendrá los horarios

correspondientes a las sesiones de esa película en ese cine. Al igual que en la anterior forma de representar los datos, al elegir uno de los horarios de una sesión, se redirige al usuario a la página web correspondiente para realizar la compra de la entrada. Se puede ver el diagrama de ejecución en la figura 3.4.

Desde cada uno de los elementos que representan a una película en la lista, se puede acceder a la representación en un mapa de la posición de los cines en relación a la del usuario. En este mapa también se permite calcular el trayecto hasta cualquiera de las ubicaciones disponibles como se indica en la figura 3.3.

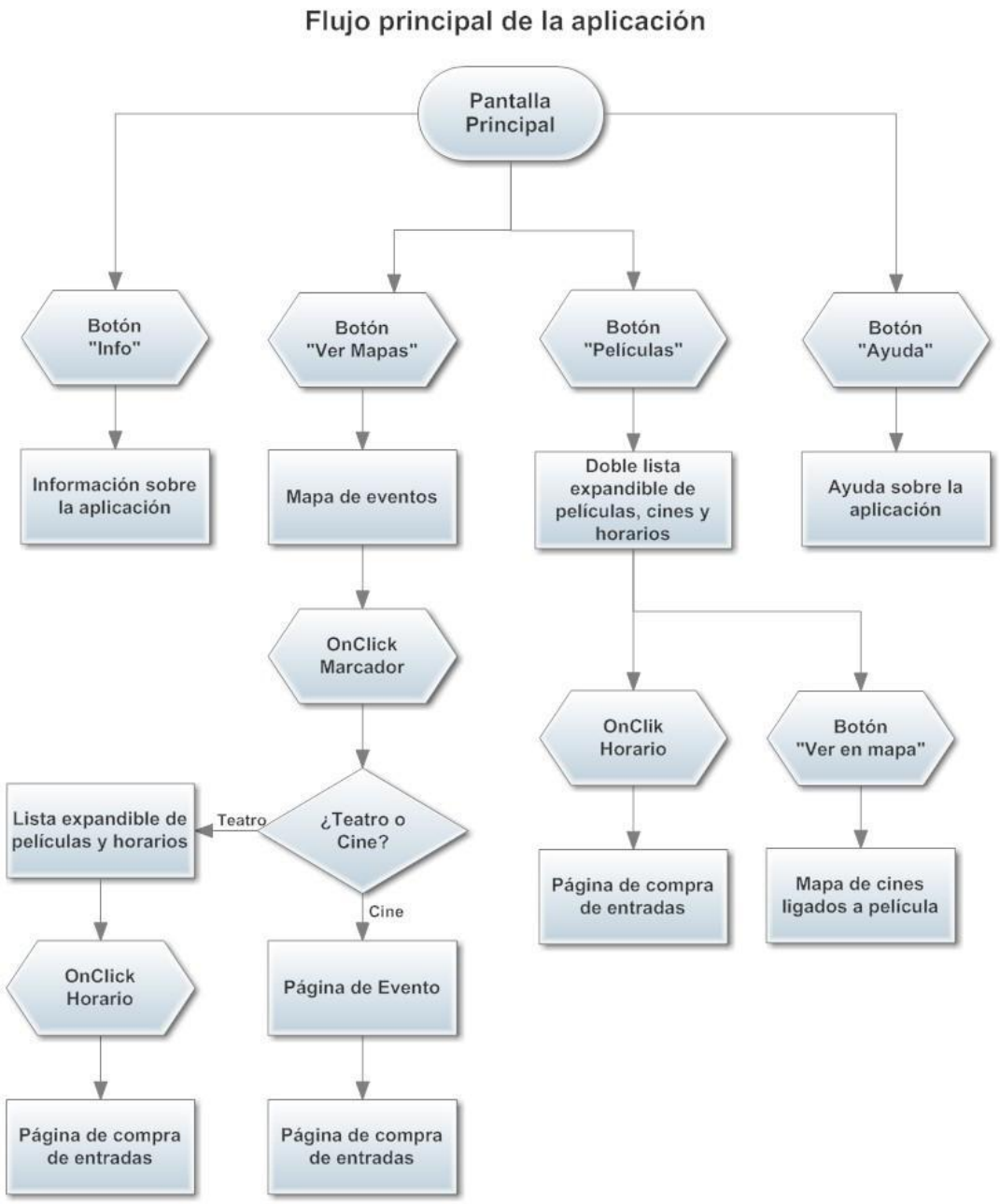

Figura 3.4 Diagrama del flujo principal de la aplicación

Desde la pantalla principal de la aplicación el usuario puede volver a lanzar la búsqueda de eventos cercanos. En este caso se realizarán las mismas acciones que al inicio de la aplicación que están reflejadas en la figura 3.1, es decir, se recalculará la posición del usuario y se volverán a ejecutar las peticiones web a los proveedores de información registrados en la aplicación. Una vez terminada la operación, se comprobará si existen eventos cercanos o no, devolviendo el control al usuario para que elija la forma en la que desea ver la información.

3.2.2 Funcionalidad de otras aplicaciones de entradas

3.2.2.1 Aplicación de entradas.com

La aplicación de entradas.com es una aplicación generalista que permite la compra de entradas para todo tipo de eventos. La búsqueda de eventos se hace a nivel de provincia, y tiene la opción de utilizar el GPS para ubicar al usuario y mostrar resultados que correspondan a la provincia en la que se encuentra. Permite el cambio manual de la provincia por si se desea ver los eventos de otras ciudades.

En la interfaz principal de usuario muestra los eventos destacados y varios buscadores para explorar los eventos disponibles. Una vez realizada una búsqueda por nombre, tipo de evento, etc. dispone de una opción para acotar los resultados a aquellos que se encuentren a una distancia máxima del usuario y descartar los demás. Además, esta distancia es configurable por el usuario.

Al escoger un evento determinado, ofrece la opción de calcular el trayecto hasta él. Para ello abre en la aplicación una ventana con un mapa de Google maps en el que se encuentra marcada la posición hasta la que se quiere llegar. Se puede elegir diferentes medios de transporte para calcular la ruta y permite elegir la situación actual como origen de ésta. En la práctica no se ha conseguido calcular el trayecto en ninguna ocasión, la aplicación se ralentiza y lo máximo que se ha conseguido es visualizar el mapa con el lugar del evento marcado.

En general la aplicación es muy lenta y muchas de las opciones no se han podido ejecutar a pesar de haber hecho pruebas en diferentes terminales con diferentes versiones del sistema operativo Android.

3.2.2.2 Ticketmaster

Esta es otra aplicación generalista para la compra de entradas para diferentes eventos. La interfaz principal es más sencilla que la de la aplicación de entradas.com, pero su respuesta es mucho mejor. En la pantalla de inicio se muestran una serie de eventos destacados y un buscador que permite filtrar la búsqueda por nombre de espectáculo o recinto, por tipo de evento, por localidad o ciudad y por fecha.

El resultado de la consulta es una lista de eventos ordenados por orden alfabético. Para cada evento muestra una descripción del mismo e indica la fecha, hora y lugar en el que se produce. Al elegir un elemento de la lista redirige al usuario a la página web correspondiente dentro del dominio de ticktackticket.com

No incluye ningún tipo de mapa asociado ni localización del usuario.

Hay que destacar que los eventos que aparecen en esta aplicación no aparecían en la anterior y viceversa.

3.2.2.3 Qhaceshoy

Qhaceshoy se define como una aplicación para improvisar planes. Se trata de una utilidad que permite la compra de última hora de entradas para diferentes tipos de evento. Se centra únicamente en eventos en vivo como obras de teatro, monólogos, musicales, conciertos, etc.

Está enfocada a la visualización de los eventos por posición, pero no utiliza los sistemas de geolocalización del terminal, en lugar de ello lista los resultados a nivel de ciudad. No está pensada para mostrar todos los eventos, únicamente incluye aquellos que se vayan a representar en las próximas horas. Utilizan un sistema comercial por el cuál invitan a los organizadores de los eventos a contactar con ellos y vender a un precio reducido las localidades que aún no se hayan vendido. De esta manera la aplicación gana usuarios y la empresa organizadora incrementa ganancias con los espectadores de última hora.

Por cada evento se muestra el tipo, género y una pequeña descripción. También muestra el precio de la entrada y ofrece la opción de comprarla directamente a través de la propia aplicación. Por último incluye un pequeño mapa con la situación del evento con la posibilidad de calcular el trayecto desde la posición del usuario hasta el lugar escogido. La aplicación no gestiona el mapa, si no que redirige la acción a una instancia de la aplicación de Google Maps con la ubicación del evento.

3.2.2.4 Atrapalo.com

Es una aplicación generalista para la búsqueda de ofertas para viajes, alquiles de coches, hoteles, entradas para eventos, etc. Tiene una función mucho más amplia que las anteriores, aunque aquí se analizará únicamente el aspecto de la venta de entradas.

Centrada la atención únicamente sobre la parte de compra de entradas para eventos, la principal diferencia con respecto a las anteriores es que no se trata de una aplicación nativa, si no de una aplicación web diseñada para teléfonos móviles.

En su interfaz principal aparece un menú con las diferentes áreas sobre las que da servicio, entre ellas la de compra de entradas. Al acceder a la parte de compra de entradas

aparece un buscador con los campos ciudad, evento, categoría y fecha. La aplicación permite el uso del GPS para obtener la ciudad o provincia en la que se encuentra el usuario (esta aplicación tampoco tiene en cuenta la posición exacta) y realiza las búsquedas únicamente en esa ciudad o provincia. También es posible introducir la localidad manualmente, pero debe coincidir con los valores previamente establecidos en la aplicación.

Una vez escogida la ciudad los resultados se muestran en una lista que se puede ordenar por eventos más vendidos, precio o más valorados. Además permite aplicar una serie de filtros para escoger por tipo de evento, por precio o por tipo de pago (para indicar si se permite el pago *on-line*).

Para cada evento se ofrece una ficha en la que se muestra el precio, una puntuación media de los usuarios, una pequeña descripción, un mapa con la situación del recinto con información sobre las líneas de transporte público cercanas y la modalidad de pago y recogida de entradas que tiene.

Esta aplicación es más completa que las anteriores y permite una mayor flexibilidad de búsqueda. Aunque de nuevo la búsqueda se hace en función de localidad o provincia y no hace uso interactivo de los mapas.

3.2.3 Comparativa de funcionalidades

En este apartado se va considerar la funcionalidad que se quiere implementar en la aplicación desarrollada en este proyecto con la funcionalidad que ofrecen las aplicaciones que se han examinado antes.

En la aplicación a desarrollar se utiliza como criterio principal de búsqueda de eventos las **coordenadas exactas** en las que se encuentra el usuario. En cambio, en todas las otras aplicaciones examinadas, la búsqueda se hace en función de la ciudad o provincia, además, en las aplicaciones de *Qhaceshoy* y *Ticketmaster* este dato debe ser introducido por el usuario. En el caso de *Entradas.com* y *Atrapalo.com* sí que permite la obtención de la posición exacta para realizar la búsqueda, pero únicamente se tiene en cuenta para obtener la provincia. Posteriormente da la oportunidad de filtrar los resultados imponiendo una distancia máxima al usuario, pero no muestra estos eventos ordenados por proximidad.

Otra característica que se quiere potenciar en esta aplicación es la **utilización de los mapas** como interfaz para el usuario, ya que le permite, de una forma visual sencilla, comprobar su situación con respecto a todos los eventos disponibles y escoger en función a ello. De nuevo se intenta dar una mayor importancia a la posición actual del usuario con respecto a la situación de los eventos. En las otras aplicaciones la forma principal de mostrar la información es mediante listas de eventos y buscadores para filtrar los datos

obtenidos. En los casos en los que se incluye un mapa para mostrar la ubicación del evento, se hace a través de una llamada a la aplicación de Maps de Google a la que se le indica la dirección de éste. Estas aplicaciones por lo tanto no hacen una

Además, otro motivo de diferenciación que se busca con esta aplicación con respecto a las demás, es **la introducción de un perfil de usuario** que permite mostrar los eventos de una forma adaptada a las preferencias del usuario y al período de tiempo en función de la hora local. De esta manera, la aplicación monitoriza y almacena el tipo de eventos que interesan al usuario y crea un perfil sobre el que basa la manera en la que se muestra la información.

Por último, mencionar que la aplicación está integrada con otros servicios ya disponibles para realizar la compra de entradas, en lugar de crear un sistema propio que podría resultar no compatible. La Tabla 1 resume esta información de acuerdo con las características identificadas.

	Entradas.com	**TicketMaster**	**QHacesHoy**	**Atrapalo**	**Aplicación desarrollada**
Localización	Por localidad	No	Por localidad	Por Localidad	Por coordenadas exactas
Integración de mapas	Llamada a aplicación Maps de Google	No	Llamada a aplicación Maps de Google	Imagen de posición	Google Maps v2 controlado por la aplicación
Integración de cálculos de rutas	No	No	Sí	No	Sí
Personalización	Filtro manual de distancia	No	No	Filtros manuales de precio, género y forma de pago	Filtro manual de género y automático basado en el perfil del usuario
Efectuar compra	Sí	Sí, redirigido	Sí	Sí	Sí, redirigido

Tabla 3.1 Comparativa ente aplicaciones de venta de entradas

3.3 Diseño de la aplicación

Para facilitar la modularidad y la escalabilidad de la aplicación, se estructura en paquetes en los que se agrupan las clases según su funcionalidad. De esta manera se pueden incluir más funcionalidades o más fuentes de información de una manera sencilla y fácilmente acoplable con el resto de la aplicación.

- Paquete principal: Este paquete lo componen las actividades que pertenecen a la ejecución del programa principal. Incluye la actividad que funciona como interfaz inicial y que contiene toda la información recogida sobre los eventos; además, permite acceder a ella de diferentes maneras. También incluye las actividades que muestran la información acerca de la aplicación y ayuda para el uso de ésta.
- Paquete de mapa: Este paquete lo conforma únicamente la actividad encargada de mostrar un mapa en el que se representan con marcadores la posición de los diferentes eventos junto con la posición del usuario.
- Paquete de eventos: Este paquete incluye las clases que empaquetan los datos relativos a los eventos. Se divide en tres carpetas:
    - Carpeta raíz: En esta carpeta se encuentra la clase que contiene toda la información recogida por la aplicación sobre los eventos. También contiene la clase que representa la unidad básica de información que se debe almacenar sobre un evento.
    - Carpeta de cines: En esta carpeta se encuentran las clases que empaquetan eventos de tipo cine.
    - Carpeta de teatros: En esta carpeta se encuentran las clases que empaquetan eventos de tipo teatro.
- Paquete de tareas: Este paquete contiene las clases que permiten la obtención de los datos relativos a los eventos. Se dividen en dos grupos:
    - Tareas: Estas clases se encargan de conectarse a la red, realizar las peticiones a los servicios web y recuperar las respuestas. Estos procesos son lentos y no se permite que se hagan en el hilo principal de ejecución, ya que lo bloquearía durante un tiempo demasiado largo. Por ello estas clases crean un hilo paralelo que se ejecute en segundo plano y que devuelva la respuesta cuando haya terminado. De este grupo se tiene la clase que obtiene datos de eventos de tipo cine de la página web de www.google.es/movies [5] y de eventos de tipo teatro del servicio web de Kulturklik [4].
    - Parseadores: Estas clases recogen las respuestas que han generado las clases anteriores y extraen los datos útiles de ellas. Estos datos se introducen en objetos java para poder manejarlos posteriormente. En este paquete se tienen dos tipos de parseadores, uno para la respuesta *jsoup* de los cines y otro para la respuesta *json* de los teatros.
- Paquete de lista de cines: Este paquete contiene las actividades que permiten representar en listas las características de los eventos de tipo cine. Esta lista muestra una lista expandible con todas las películas que se proyectan en el cine elegido. Los elementos de la lista que representan a las películas contienen a su

vez otra lista, que se expande cuando el usuario presiona dicho elemento, con los horarios de las sesiones para esa película. Al presionar sobre un horario se ofrece la opción de ir a la página correspondiente a comprar la entrada para esa sesión.

- Paquete de lista de películas: Paquete con las clases que permiten ver la información de los eventos de tipo cine por películas. En este caso se crea una lista expandible de dos niveles. En el nivel principal se encontrará una lista con todas las películas que se proyectan en los cines cercanos al usuario. Las películas están ordenadas según las preferencias del usuario con respecto al género y según la importancia de la película en la cartelera (se mostrarán primero los estrenos, después las más taquilleras, etc.). Cada entrada de la lista de películas contendrá un botón para poder mostrar en un mapa la posición de los cines que proyectan esa película. En el paquete se incluye a actividad que representa a la lista, los adaptadores de la lista para poder gestionar los dos niveles y la actividad que muestra el mapa.
- Paquete de rutas: En este paquete se encuentran todas las clases que permiten representar las rutas en el mapa. Se divide en dos paquetes:
    - Paquete de elementos pasivos: Este paquete incluye las clases que empaquetan los datos obtenidos desde el servicio web.
    - Paquete de elementos activos: Este paquete incluye las clases que gestionan la conexión con la red y el paso de los datos a las clases del primer paquete. Como en el caso de la obtención de datos sobre eventos, la conexión a Internet ha de hacerse sobre una clase que cree un hilo secundario para ejecutarse en segundo plano. Este hilo enviará la petición y recogerá la respuesta al servicio web de Google Directions. Después habrá otra clase que parsee la respuesta *json* e introduzca la información necesaria en las clases del paquete de elementos pasivos. Por último se incluye una actividad que permita visualizar los detalles de la ruta.

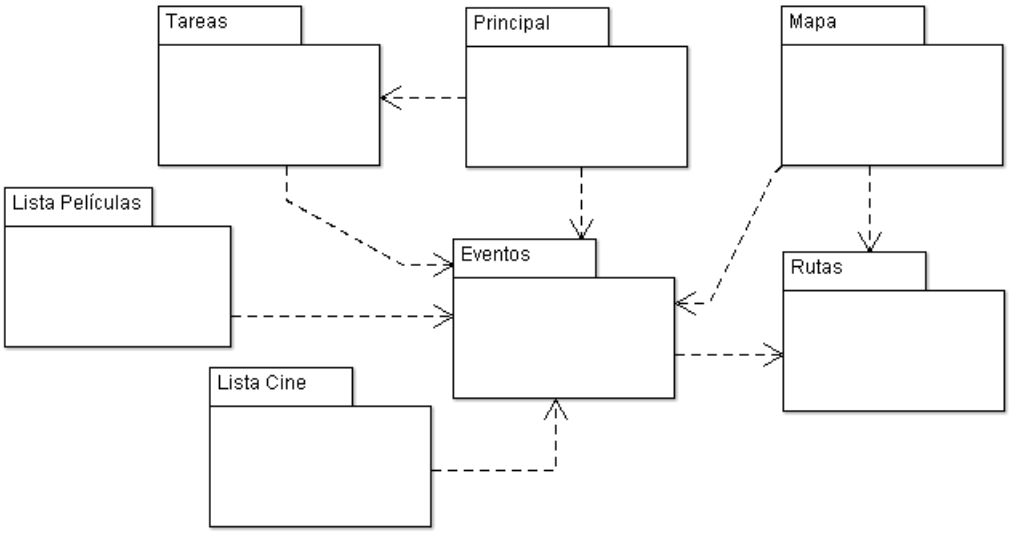

Figura 3.5 Diagrama de paquetes de la aplicación

3.4 Especificación de requisitos de software

3.4.1. Requisitos funcionales

- La interfaz de la aplicación debe ser clara y sencilla para facilitar su uso.
- La aplicación debe sugerir al usuario la activación del GPS en el caso de que se encuentre desactivado.
- La aplicación debe de seleccionar el mejor sistema disponible en cada situación para obtener la geoposición del usuario.
- La aplicación debe obtener la geoposición del usuario para utilizarla como criterio de búsqueda de datos.
- La aplicación debe proporcionar al usuario la posibilidad de comprar entradas para eventos que se encuentren cerca de su posición.
- La aplicación debe poder guiar al usuario hasta las pasarelas de pago para la compra de las entradas.
- La aplicación debe permitir al usuario decidir sobre el tipo de eventos sobre los que quiere información.
- La aplicación debe de ser capaz de extraer los datos referentes a la cartelera de los cines que se encuentren cerca de la posición del usuario.
- La aplicación debe de ser capaz de contacta con servidores web disponibles y extraer los datos referentes a las obras que se representan en los teatros cercanos al usuario.
- La aplicación debe de poder representar la información obtenida y de localizarla en un mapa.
- La aplicación debe de poder facilitar al usuario el trayecto que se ha de hacer para llegar desde su posición actual hasta el lugar en el que se representa el evento al que desea acudir.
- La aplicación debe de mostrar la información de un evento directamente si el usuario se encuentra a una distancia mínima de la posición del evento.
- La aplicación debe de guardar un perfil del usuario en el que se almacenen sus preferencias en cuanto al tipo de búsqueda, tipo representación de mapa y tipo de género de eventos
- La aplicación debe mostrar la información obtenida adaptada a las preferencias de cada usuario.

3.4.2 Requisitos no funcionales

- El dispositivo móvil debe tener conexión a Internet.
- El dispositivo móvil debe tener receptor GPS u otro tipo de sistema de geolocalización.
- El usuario debe tener instalado en el dispositivo el servicio de Google Play.

- El usuario debe tener una cuenta de Google activa en el dispositivo móvil.
- El dispositivo móvil debe tener instalado el sistema operativo Android con una versión 2.2 o superior.

3.5 Casos de uso

La aplicación está pensada para facilitar la compra de entradas de diferentes eventos desde un dispositivo móvil. El usuario puede utilizarlo en dos tipos de escenarios:

En un primer escenario, un usuario con un dispositivo móvil con acceso a Internet se encuentra dando un paseo o tomando algo en algún establecimiento y está cavilando sobre lo que puede hacer a continuación:

1. Por medio de la aplicación puede consultar en su terminal los eventos que se están representando cerca de su posición.
2. El usuario puede explorar el mapa para ver la situación relativa de los eventos con respecto a la suya y explorar las características de cada uno de ellos.
3. El usuario puede ver el trayecto que ha de seguir para llegar desde su posición actual hasta el lugar en dónde se representa el evento elegido. El usuario tendrá acceso además a las características de la ruta en el que se le indica la distancia, el tiempo estimado que se tarda en recorrerla y las indicaciones escritas paso a paso.
4. Una vez que ha escogido un evento en particular el usuario puede pagar y sacar la entrada a través de su terminal.
5. El usuario llega hasta el lugar del evento y accede directamente sin pasar por la taquilla.

En un segundo escenario, un usuario con un dispositivo móvil con conexión a Internet se encuentra en la puerta del evento que desea ver y está en la cola de la taquilla esperando para comprar su entrada:

1. El usuario consulta la aplicación para obtener la información sobre los eventos que se encuentran cerca de su posición.
2. La aplicación detecta que el usuario se encuentra en los alrededores del lugar del evento en el que está esperando en la cola y muestra por pantalla la información disponible del evento.
3. El usuario selecciona el evento concreto y va a la página web que le permite comprar la entrada a través del dispositivo.
4. El usuario puede salir de la cola de taquilla y acceder directamente al evento.

Capítulo 4. Implementación

En el capítulo anterior se vio el diseño y funcionalidad requerida para la aplicación a desarrollar en este proyecto. En este capítulo se va a explicar de forma detallada las clases que la componen y su implementación para alcanzar los requisitos funcionales.

En el capítulo anterior se ha utilizado el nombre de clase para definir a todos los componentes java. A partir de ahora vamos a distinguir un tipo especial de clase del resto. Este tipo de clase va a ser la actividad. Llamaremos actividad a una clase que extienda de la clase Activity directa o indirectamente. La clase Activity es una clase del paquete de android cuya principal particularidad es que tiene asociado una interfaz con la que el usuario puede interactuar y controlar el flujo de la aplicación.

Para cada actividad se va a mostrar el diagrama de clases asociado a ella.

4.1 Paquete principal

Este es el paquete principal de la aplicación. Contiene la actividad principal Kinetea, que es el interfaz principal a partir de la cual se accede a todas las funcionalidades de la aplicación. Además, en este paquete también están las actividades que muestran información de la aplicación y una pequeña guía de usuario.

4.1.1 Actividad principal, Kinetea

La clase principal de la aplicación es la clase Kinetea. Extiende de la clase Activity y por lo tanto ofrece una interfaz al usuario, e implementa la interfaz LocationListener para recibir la posición a través de los sistemas de geolocalización [28][29].

Esta actividad no se destruirá durante toda la ejecución de la aplicación y almacenará todos los datos referentes a la posición de usuario y a los eventos encontrados. Será la encargada de interactuar con el usuario para derivarle a todas las opciones de la aplicación.

4.1.1.1 Interfaz

Esta clase es la interfaz gráfica principal del usuario. La interfaz se compone de cuatro botones como se puede observar en la figura 4.1:

- Botón Actualizar: Al pulsar este botón la aplicación vuelve a utilizar el sistema de geolocalización para actualizar su posición y con ella conectarse a los proveedores de datos para obtener toda la información sobre los eventos que se encuentren en las proximidades del usuario. Este proceso ya se hace automáticamente al iniciarse la aplicación. Este botón permite refrescar los datos.
- Botón Ver Mapa: Lanza la actividad Mapa que representa la posición actual del usuario en un mapa junto a las posiciones de los eventos que se han obtenido antes.
- Botón Ver Películas: Lanza la actividad ListaPeliculas que muestra una lista expandible doble con todas las películas que se están proyectando en los cines de los cuales hemos obtenido la información. Esta lista vendrá ordenada basándose en las preferencias del usuario.
- Botón Menú: Muestra el menú desplegable de la aplicación con tres opciones:
    - Ayuda: Muestra la actividad Ayuda en la que se explica brevemente la funcionalidad de la aplicación en formato texto.
    - Info: Muestra la actividad Info en la que se ofrecen datos sobre la aplicación y su autor.
    - Tipo de búsqueda: Permite al usuario decidir sobre qué tipo de eventos se va a buscar información. La elección se realizará mediante un menú emergente que contendrá las opciones posibles seleccionables. En esta primera versión de la aplicación las entradas del menú son teatros, cines o ambos.
- Cuadro de texto de información de progreso: Cuadro de texto que informa al usuario de la situación del progreso de la tarea que obtiene la información de la red y que se ejecuta en segundo plano. Será visible únicamente cuando se esté ejecutando dicha tarea.
- Barra de información de progreso: Barra que muestra el porcentaje de datos que se han recuperado en la tarea que se ejecuta en segundo plano. Será visible únicamente cuando se esté ejecutando dicha tarea.

Figura 4.1 Interfaz principal de la aplicación

Figura 4.2 Menú desplegado en la interfaz principal

4.1.1.2 Atributos

- LocationManager locationManager: Este atributo instancia la clase LocationProvider que permite interactuar con los sistemas de localización para obtener la posición del usuario, configurar el tiempo o la distancia de refresco del dato o para ejecutar métodos cuando el usuario esté cerca de una ubicación determinada. Para instanciar el objeto, se obtiene de las propiedades del sistema: [30]

```
locationManager = (LocationManager)
      getSystemService(Context.LOCATION_SERVICE);
```

- String provider: En esta cadena de texto se almacena el tipo de sistema de localización que se va a emplear. Para instanciarlo lo hacemos escogiendo el más adecuado de los disponibles a través del LocationManager. Para ello llamamos al método getBestProvider():

```
provider = locationManager.getBestProvider(new Criteria(), false);
```

- Eventos cartel: Este es el objeto que contiene toda la información sobre los eventos. Lo asociamos con la actividad principal para poder ser referido en cualquier parte de la aplicación. Tendrá un carácter estático.

- double latitud, longitud: Almacenamos las coordenadas de la posición del usuario.

- Geocoder geoCoder: Instanciamos un objeto de la clase Geocoder para poder acceder a través de él a las funcionalidades de geocodificación y geocodificación inversa. En realidad, aunque en el código se ejecute como una llamada a un método estático, se está ejecutando una consulta a un servicio web, pero está encapsulado en android para poder utilizarlo como una clase normal [31].

- SharedPreferences preferences: Creamos una instancia de la clase SharedPreferences para leer y sobrescribir las opciones del usuario y sus preferencias. De esta forma podemos guardar las elecciones del usuario entre sesiones [32].

4.1.1.3 Métodos

- onCreate(Bundle): Este método sobrescribe el del mismo nombre de la clase Activity. Se ejecuta cuando se accede a la aplicación por primera vez. Inicializa los objetos de sesión que se van a utilizar durante todo el ciclo de vida de la aplicación.

Lo primero que hace el método es llamar al método onCreate() de la clase padre. Después asocia la interfaz gráfica definida en kinetea_main.xml a la actividad, crea los objetos del LocationManager, Geocoder y SharedPreferences, y escoge el mejor sistema para obtener la localización.

Inicialmente el sistema de localización que permite una mayor precisión es el GPS, por lo que el método comprueba si está activado y en el caso negativo, pregunta al usuario si desea activarlo o no. Si el usuario desea activarlo se le redirige a la pantalla de opciones del sistema en dónde puede activarlo. El sistema operativo no permite la activación automática de este sistema.

Una vez definido el proveedor de geolocalización, se obtiene la posición del usuario y se almacena. A continuación se asocian escuchadores a los botones de Actualizar, Ver Mapa y Ver Películas del interfaz y se definen las acciones que van a realizar cuando el usuario interactúe con ellos. Por último crea una instancia de TareasCartelera e inicia otro hilo con la búsqueda de información de los eventos.

- onResume(): Este método sobrescribe el del mismo nombre de la clase Activity. Se ejecuta cuando una actividad cualquiera empieza a estar disponible para el usuario y se encuentra en lo alto de la pila de actividades. Esto sucede cuando se crea dicha actividad por primera vez y cuando se vuelve a ella después de una pausa.

Al sobrescribirlo la primera sentencia es la llamada al método sobrescrito de la clase padre. Después invoca al método requestLocationUpdates() del LocationManager para que envíe periódicamente la posición del usuario a la aplicación.

- onPause(): Este método sobrescribe el del mismo nombre de la clase Activity. Se ejecuta en una actividad cuando se lanza otra que recibirá el control del usuario y por lo tanto se situará encima de ella en la pila de procesos. Se ejecuta antes de pasar el control a la nueva actividad.

Al sobrescribirlo se ejecuta primero el método sobrescrito de la clase padre. Después se invoca al método removeUpdates() de la clase LocationManager para que deje de enviar periódicamente la posición del usuario a la aplicación.

- onCreateOptionsMenu(Menu): Este método sobrescribe el del mismo nombre de la clase Activity. Se utiliza para inicializar el contenido del menú de opciones desplegable. En él indicamos el archivo kinetea_menu.xml en el que se describe el formato y apariencia del menú de la aplicación.

- onOptionsItemSelected(MenuItem): Este método sobrescribe el del mismo nombre de la clase Activity. Controla la selección de cada una de las opciones disponibles en el menú desplegable de la aplicación.

Se implementa mediante un switch - case utilizando como condición el identificador del elemento sobre el que se ha pulsado y que se pasa como parámetro del método. Tanto las interacciones con objetos del menú como las de los submenús se controlan en este método de la misma forma y se encuentran al mismo nivel lógico dentro de la estructura switch - case.

- onLocationChanged(Location): Este método se ha de declarar al implementar la interfaz LocationListener. Se ejecuta cuando la posición del usuario cambia siguiendo el criterio que se le ha especificado al LocationManager. Recibe la posición de usuario y se asignan las coordenadas de la nueva posición a los atributos de latitud y longitud de la clase Kinetea.

- onProviderDisabled(String), onProviderEnabled(String), onStatusChanged(String, int, Bundle): Estos métodos se han de declarar al implementar la interfaz LocationListener, pero en no se utilizan y por lo tanto están vacíos.

4.1.1.4 Diagrama de clases

Esta actividad contiene un objeto de la clase Eventos que, como se verá más adelante, es la clase que almacena toda la información que se extrae de la red y contiene listas de los objetos que representan los cines y los teatros. Además esta actividad es la encargada de lanzar la tarea secundaria que realiza la obtención de los datos y los almacena en sus correspondientes objetos:

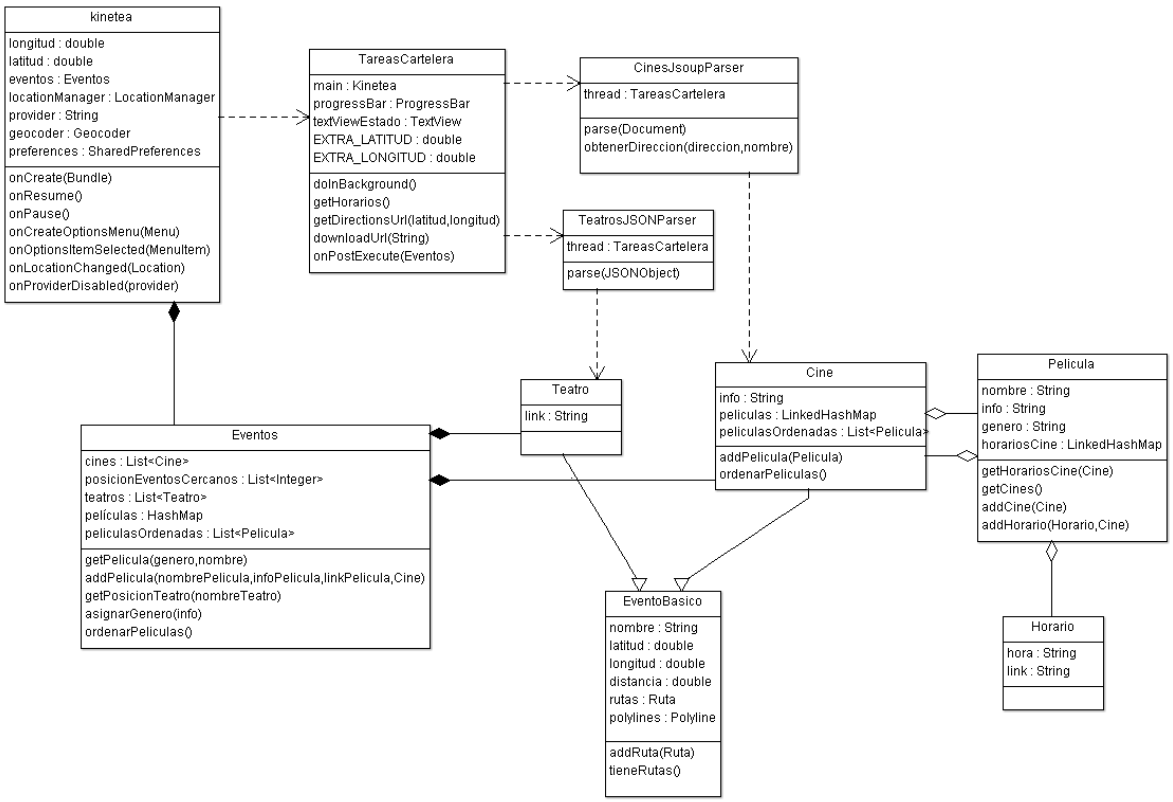

Figura 4.3 Diagrama de clases dependientes de la actividad Kinetea

4.2 Paquete **Tareas**

En este paquete se encuentran las clases que van a permitir conectarse a la red para extraer la información que nos interesa y de encapsularla en objetos de las clases del paquete de eventos. La clase TareasCartelera creará un nuevo hilo de ejecución para conectarse a Internet de dónde obtendrá los datos de interés y utilizará las clases CinesJSONParser y TeatrosJSONParser para empaquetarlos en sus correspondientes objetos.

4.2.1 **TareasCartelera**

Esta clase extiende de la clase AsyncTask que permite realizar operaciones en segundo plano en un nuevo hilo diferente al principal e informar a éste de su progreso. Las conexiones a la red para obtener datos pueden ralentizar la ejecución del hilo principal en exceso y causaría la ralentización de todas las demás funcionalidades del terminal, por ello sólo están permitidas en tareas en hilos paralelos. AsyncTask ofrece una forma sencilla de hacerlo. Los métodos que define esta clase son [33]:

- onPreExecute(): Se ejecuta en el hilo principal antes de que se empiece a ejecutar la tarea en segundo plano

- doInBackground(Params...): Se ejecuta inmediatamente después de onPreExecute(). En este paso se realiza la tarea principal del hilo.
- onProgressUpdate(Progress...): Se ejecuta después de que se haga una llamada a publishProgress(Progress...) durante la ejecución de la tarea principal y se suele utilizar para ofrecer información acerca de la progresión de esta tarea.
- onPostExecute(Result): Se ejecuta después del método doInBackground() y recibe como parámetro de entrada la salida de éste.

4.2.1.1 Atributos

- Kinetea main: Guardamos una instancia de la actividad principal para poder acceder a la interfaz y habilitar / deshabilitar los botones de la interfaz.

- ProgressBar progressbar: Referencia a la barra de progreso de la interfaz principal que muestra al usuario el porcentaje de datos que quedan por cargarse.

- TextView textViewEstado: Referencia al cuadro de texto de la interfaz principal que aparece cuando se está ejecutando la tarea e informa al usuario primero, de que se está conectando a Internet y después, que se está obteniendo la información.

- double EXTRA_LATITUD: Para definir la región en la que se busca información en el servicio web de Kulturklik, se necesita especificar unas latitudes y longitudes máximas y mínimas. He introducido los valores como constantes que se pueden modificar para ajustar o ampliar la búsqueda. Este valor es lo que se suma y resta a la posición de usuario para obtener la latitud máxima y mínima respectivamente.

- double EXTRA_LONGITUD: Este valor es lo que se suma y resta a la posición de usuario para obtener la longitud máxima y mínima respectivamente.

4.2.1.2 Métodos

- Constructor: El constructor recibe como parámetro una referencia a la actividad principal Kinetea para poder realizar cambios en la interfaz mientras se ejecuta la carga de datos.

- doInBackground(): Este método sobrescribe al método del mismo nombre de la clase AsyncTask. Este método es el método principal que se ejecuta cuando se crea una tarea asíncrona.

En primer lugar utilizamos la referencia a la clase principal para hacer invisibles los botones de la interfaz principal para que el usuario no pueda interrumpir la tarea que se está ejecutando lanzando otro tipo de acciones. En lugar de los botones aparecerá un

cuadro de texto indicando que la aplicación se está intentando conectar a Internet para obtener los datos.

A continuación se consulta en las preferencias de la aplicación el tipo de búsqueda especificada por el usuario (recordemos que estas preferencias se guardan de una sesión para otra). Según este valor se ejecutarán los métodos de obtención de los datos de los cines (getHorarios()), de los teatros (getTeatros()) o de ambos.

-getHorarios(): Este método se encarga de conectarse a internet y obtener la información relativa a las sesiones de los cines que se encuentran cerca de la posición del usuario de la página de www.google.es/movies.

Primero crea la *url* correspondiente a la cartelera completa para el día actual introduciendo la información de localización almacenada en la actividad principal. Después se utiliza la librería de *jsoup* para extraer toda la información contenida en dicha página web con el método estático connect() y se almacena en un objeto del tipo Document [24].

Esta conexión para descargar la información de la página web es mucho más lenta que la conexión a un servicio web, por ello mientras se realiza se muestra el mensaje Conectando en la pantalla de la actividad principal. Una vez se ha terminado la descarga de los datos, se cambia el texto del mensaje por el de Obteniendo información y se activa la barra de progreso que informará al usuario de que los datos se están almacenando. Después se llama al método parse(Document) de la clase CinesJsoupParser en dónde se extraerán los datos de interés de la página y se cargarán en la clase Eventos.

- getTeatros(): Este método se encarga de conectarse al servicio web de Kulturklik para obtener las obras de teatro que se están representando cerca de la posición de usuario.

Primero se obtiene la *url* que tenemos que formar a partir de la ubicación del usuario con el método getDirectionsUrl(latitud,longitud). Realizamos la petición a través del método downloadUrl(url) que nos devolverá un texto con la respuesta del servidor. Esta respuesta en formato de texto plano, se trata como un objeto JSONObject que se parsea con el método parse(JSONObject) de la clase TeatrosJSONParser y carga los datos en la clase Eventos.

- getDirectionsUrl(Double,Double): Este método se encarga de crear la url que sirva como petición del servicio web Kulturklik. Recibe como parámetros la posición del usuario y devuelve la *url* necesaria para realizar la petición.

A partir de la posición de usuario y con los valores constantes especificados en la aplicación, se calcula la región sobre la cual se va a realizar la búsqueda. Suma y resta los valores de latitud y longitud de la posición del usuario a las constantes definidas y se

obtienen las coordenadas máximas y mínimas. Se introducen el formato requerido y devuelve la *url* formada.

- downloadUrl(String): Este método se conecta al servicio web Kulturklik y extrae la información de los eventos que están cerca de la posición del usuario. Recibe como parámetro la *url* a la que se ha de conectar y devuelve la respuesta como una cadena de texto.

Crea un objeto URL con la dirección especificada y realiza la conexión HTTP. Después crea un flujo de datos para leer la respuesta y almacenarla.

- onProgressUpdate(Integer): Este método sobrescribe al método de la clase AsyncTask. Se ejecuta cuando en el hilo principal se llama a publicProgress en el cual se devuelve el parámetro que utiliza de entrada el primer método. Se utiliza como una forma de informar al hilo principal de cómo va el proceso asíncrono.

Actualiza el porcentaje de obtención de datos en la barra de progreso mostrada en la actividad principal.

- onPostExecute(Eventos): Este método sobrescribe al de la clase AsyncTask. Se ejecuta cuando la tarea que estaba realizando el método doInBackground() termina y como parámetro de entrada recibe el elemento que ha devuelto este último.

Asigna el objeto Eventos que se ha creado al menú principal para que pueda ser consultado desde cualquier punto de la aplicación. También devuelve el control a la interfaz principal ocultando el cuadro de texto y la barra de progreso que informaban sobre la ejecución de la tarea en segundo plano.

4.2.2 CinesJsoupParser

Esta clase se encarga de parsear la información que extraemos de la web de www.google.es/movies introduciendo los datos en objetos propios del paquete de eventos que nos permitirán después interactuar con ellos.

4.2.2.1 Atributos

- TareasCartelera thread: Guardamos una referencia al hilo que llama a la clase para tener acceso a sus métodos.

4.2.2.2 Métodos

- parse(Document): Este método es el que se encarga de hacer la función principal de la clase. Recibe como parámetro el objeto Document que contiene la información de los eventos de tipo cine y la extrae para encapsularla en objetos del paquete de eventos.

Como se vio en el capítulo 2, toda la información que necesitamos de la página web de cines de Google se encuentra encapsulada dentro de un tag div perteneciente a la clase theater. Extraemos este grupo del objeto Document con el método select(String pattern), pasándole como parámetro div[class^=theater] y obteniendo un objeto de la clase Elements. Este objeto contiene una lista de objetos Element que comprende toda la información relacionada con cada uno de los cines. Cada objeto Element relacionado con un cine contiene objetos de este tipo relacionados con cada una de las películas y a su vez, cada una de las películas contiene objetos relacionados con los horarios de reproducción de esas películas.

De esta manera, el método recorre todos los cines obteniendo sus características. Para cada cine recorre todas las películas que tiene en cartelera obteniendo también sus características y vuelve a hacer lo mismo para cada horario.

Entre las características del cine, como se verá luego, se encuentran su latitud, su longitud y su distancia al usuario. Esta información no está disponible en la página web así que se llama al método obtenerDireccion(info, nombre) para conseguirla. Según se van obteniendo los diferentes objetos se van añadiendo al objeto Eventos.

Una vez que se ha obtenido la dirección del cine se comprueba si se encuentra a una distancia mínima establecida del usuario y si es así, se añade a una lista de eventos cercanos.

Para que el usuario tenga una idea de cómo va la ejecución de la extracción de datos en la interfaz principal, se utiliza el método publicarProgreso(int) de la clase TareasCartelera. Para mostrar el porcentaje de proceso que se ha ejecutado ya, se obtiene el número de cines que se han obtenido de la búsqueda. Dependiendo de si la búsqueda se hace únicamente por cines o por cines y teatros, se divide 100 ó 50 entre el número de cines, para calcular el porcentaje que debe de avanzar la barra de progreso en cada caso. Cada vez que procesemos un cine, se añade esta cantidad al total del progreso y se actualiza la barra de la actividad principal.

- obtenerDireccion(direccion, nombre): Este método se encarga de obtener las coordenadas y la distancia del usuario hasta un cine. Recibe como parámetros la información y el nombre del cine que se ha extraído de la página. Devuelve un *array* de tres elementos con la latitud, longitud y distancia.

El formato de la información del cine suele ser dirección – teléfono, aunque el teléfono puede aparecer o no. En primer lugar, por lo tanto, se extrae la dirección de la

información. Esta dirección se pasa como parámetro del método de geolocalización inversa de la clase Geocoder, por ello adaptamos el formato de la dirección para que cumpla en la mayor medida posible las indicaciones de formato de este método.

Las direcciones extraídas no siempre cumplen el formato requerido y no es posible obtener una dirección real. Por ello se toman dos medidas:

- Limitar el espacio de búsqueda de esta dirección para que el Geocoder no devuelva direcciones similares en otros lugares.
- Buscar la dirección del cine por el nombre. Muchos de los cines están registrados por Google y con su nombre se puede extraer su geoposición.

De esta manera se pasa como parámetro del método de Geocoder unas latitudes y longitudes máximas y mínimas y se consulta tanto por la dirección extraída como por el nombre. Una vez se tienen las dos direcciones se comprueba si ambas son válidas, si sólo lo es una, se utiliza esa; si son las dos válidas, se comprueba la distancia al usuario y se escoge la menor; si no lo es ninguna se devuelve una dirección con distancia 0 que se hará que el cine se descarte y no se introduzca en el objeto de Eventos. La distancia hasta el usuario se hace mediante el método estático de la clase Location distanceBetween() que recibe como parámetros las coordenadas del lugar de origen y del de destino y la referencia a un *array* de tipo float en el que devolverá los resultados.

4.2.3 TeatrosJSONParser

Esta clase se encarga de parsear la información que extraemos del servicio web de Kulturkilk introduciendo los datos en objetos propios del paquete de eventos que nos permitirán después interactuar con ellos.

4.2.2.1 Atributos

- TareasCartelera thread: Guardamos una referencia al hilo que llama a la clase para tener acceso a sus métodos.

4.2.2.2 Métodos

- parse(JSONObject): Este método es el que se encarga de hacer la función principal de la clase. Recibe como parámetro el objeto JSONObject que contiene la información de los eventos de tipo teatro y la extrae para encapsularla en objetos del paquete de eventos [34].

Se obtienen todos los eventos en un objeto JSONArray a partir del JSONObject que ha devuelto la consulta. Calculamos el número de elementos que existen y se calcula el incremento que se ha de hacer en la barra de progreso para cada uno [35].

Por cada elemento del array de JSON se comprueba el tipo de evento que es, si es de tipo teatro, se leen sus características, se calcula la distancia a la posición de usuario con el método distanceBetween() de la clase Location y se añade al objeto Eventos.

4.3 Paquete de eventos

Este paquete contiene las clases necesarias para almacenar toda la información que extraemos de la red sobre los eventos. Contiene un paquete raíz y otros dos que contendrán información más detallada de cada tipo de evento:

- Paquete raíz: En él que se encuentran la clase Eventos en el que se incluyen todas las demás y la clase EventoBasico que describe el elemento básico de evento.
- Paquete cine: En él se encuentran las clases relacionadas con un evento de tipo cine: Cine, Pelicula y Horario
- Paquete teatro: En él se encuentran las clases relacionadas con un evento de tipo teatro: Teatro

4.3.1 Eventos

Esta clase contiene toda la información acerca de los eventos que hemos obtenido a través de los proveedores de información. En ella se almacenan todos los datos y se utiliza como un atributo estático de la clase principal para que pueda ser accedida desde cualquier parte de la aplicación.

4.3.1.1 Atributos

- List<Cine> cines: Este atributo contiene una lista de elementos del tipo Cine que representan a los cines que se encuentran alrededor de la posición de usuario.

- List<Integer> posicionEventosCercanos: Este atributo contiene una lista de elementos del tipo Cine que representan a los cines que se encuentran en a menos de una distancia establecida del usuario. Estos eventos se mostrarán directamente ya que el usuario debe de estar muy cerca de ellos.

- List<Teatro> teatros: Este atributo contiene una lista de elementos del tipo Teatro que representan a los teatros que se encuentran alrededor de la posición de usuario.

- HashMap<String, LinkedHashMap<String, Pelicula>> películas: Este atributo guarda todas las películas disponibles en la cartelera y se hace un primer mapeo por género. Después se realiza un segundo mapeo, dentro de cada género, por título de película. Para este segundo mapeo se utiliza un LinkedHashMap que además de asociar el objeto con la película, guarda el orden en el que se han añadido. Esto se hace porque, al extraer de la web las películas, vienen ordenadas por importancia, colocándose los estrenos y títulos más importantes al principio [36][37].

- List<Pelicula> peliculasOrdenadas: Este atributo guarda una lista simple con todas las películas ya ordenadas. Para la ordenación se tiene en cuenta los géneros que más le interesan al usuario y dentro de cada género se ordenan por importancia y por nombre, ya que una misma película puede aparecer en su versión normal, digital, 3D, VOS, etc.

4.3.1.2 Métodos

- getPelicula(genero,nombre): Este método recibe como parámetros el género y el nombre de la película a buscar y devuelve dicha película si existe o *null* en caso contrario. La búsqueda es muy sencilla ya que el primer mapeo en el atributo películas se hace por género y el segundo por nombre.

- addPelicula(nombrePelicula,infoPelicula,linkPelicula,Cine): Este método añade una nueva película al mapa películas. Recibe como parámetros el nombre, información y enlace de la película y el cine en el que se está proyectando. Devuelve la referencia a la película que se acaba de añadir.

A partir del campo de información que se ha recuperado de la página web, se utiliza el método asignarGenero(infoPelicula) para extraer de él el género al que pertenece la película. Una vez se ha clasificado a la película se comprueba si ya se existe la entrada en el mapa de películas para ese género. Si no existe se crea el mapa, el objeto película nuevo y se añade; si existe, se comprueba que no exista ya una entrada con el mismo nombre en el segundo mapa. Si no existe una entrada con el nombre de la película, se crea el objeto película y se añade; si existe, se obtiene una referencia. En todos los casos se guarda la película para añadirle el objeto Cine y devolverla al finalizar el método.

- getPosicionTeatro(nombreTeatro): Este método busca en el listado de teatros un teatro con el nombre dado como parámetro. Si lo encuentra devuelve la referencia al objeto y si no, devuelve *null*.

- asignarGenero(info): Este método extrae la el género de la película de la información general. El formato de información de una película puede contener o no varios campos separados por guiones. Al no mantener una posición fija ni contener siempre el mismo

número de campos, buscamos iterativamente si la información contiene las cadenas de texto que identifican su género.

- ordenarPeliculas(): Este método ordena la lista de películas para mostrarla al usuario. Devuelve el listado de las películas ordenadas.

Lo primero que se comprueba es si ya se ha creado anteriormente la lista ordenada porque ya se haya llamado previamente al método. Si es así se devuelve la lista y si no es así se calcula y se almacena en el atributo peliculasOrdenadas de la clase.

Primero se ordenan por géneros dependiendo de las preferencias del usuario. Cada vez que el usuario decide comprar la entrada de una película, añade un punto al *ranking* del género al que pertenezca dicha película. El *ranking* de cada género se almacena en el objeto de SharedPreferences de la actividad principal.

Para conseguir el orden deseado, se consulta el ranking de cada género y se guarda en un mapa que asigna a cada posible valor de género el número de *ranking* que le corresponde. Se extrae un ArrayList del mapa con el nombre de los géneros y se ordena utilizando como criterio su ranking.

Una vez ordenados los géneros, se obtienen una a una las listas de películas almacenadas en el atributo películas y se recorren para añadirlas a la lista ordenada que se devolverá como salida.

Además de las preferencias por género y de mantener el orden de importancia dado por el proveedor de datos, se ordenan las películas también por título. Una misma película puede aparecer en diferentes formatos: normal, digital, 3D, etc. y no suelen ordenarse de forma contigua. Para implementar la ordenación por títulos, se recorre cada una de las listas de películas de cada género. Para cada elemento de la lista se comprueba que no se haya añadido con anterioridad. Si no se ha añadido se añade la película y se busca hacia adelante en la lista aquellas películas que tengan el mismo título, si se encuentra alguna se añade a la lista.

4.3.2 EventoBasico

Esta clase representa el elemento básico para un evento. De él heredarán los demás tipos de eventos que se añadan a la aplicación.

4.3.2.1 Atributos

- String nombre: Nombre que define al evento.
- double latitude, longitud: Coordenadas del lugar del evento.
- String link: Enlace del evento.

- double distancia: Distancia desde el lugar del evento hasta la posición del origen.
- Ruta[] rutas: Rutas para mostrar al usuario como llegar desde su posición hasta la posición del evento. Se almacenan en un array de tres posiciones que corresponden a los tres tipos de medios de transporte que se contemplan en la aplicación: en coche, andando y en transporte público.
- Polyline[] polylines: Información para pintar las rutas del atributo anterior en el mapa. También es un *array* de tres posiciones que corresponden a los diferentes medios de transporte [38].

4.3.2 Métodos

- Constructor: El constructor recibe como parámetros los valores relativos a sus características y se inicializan los *arrays* de rutas y polylines.

- addRuta(Ruta): Obtiene de la información de la ruta que se pasa como parámetro el tipo de transporte que se ha utilizado para calcularla y se añade en la posición del *array* correspondiente.

- tieneRutas(): Este método comprueba si ya se ha calculado alguna de las rutas posibles para este evento. Devuelve true en caso afirmativo, false en caso contrario.

4.3.3 Cine

Esta clase encapsula toda la información de un evento de tipo cine. Extiende de la clase EventoBasico y hereda todos sus atributos y métodos.

4.3.3.1 Atributos

- String info: Este atributo almacena el campo información sobre el cine que obtenemos del proveedor de datos (dirección, teléfono, etc.)

- LinkedHashMap<String, List<Pelicula>> peliculas: Este atributo almacena un mapa con el listado de películas asociado por género.

- List<Pelicula> peliculasOrdenadas: Este atributo almacena un listado con las películas ordenadas según las preferencias del usuario y su importancia.

4.3.3.2 Métodos

- Constructor: Este método recibe como parámetros valores para todas las características del cine. Se llama al constructor de la clase padre para los atributos comunes (nombre, latitud, longitud, enlace, distancia), se asigna el valor de información y se inicializa el mapa de películas.

- addPelicula(Pelicula): Este método almacena la película que se pasa como parámetro en el mapa de películas.

Primero obtiene el género de la película que se quiere añadir. Comprueba si ya existe una lista creada para ese tipo de género, si no existe, la crea y añade la película al atributo películas.

Este método es más sencillo que su homólogo de la clase Eventos, ya que en este caso no hace falta comprobar si la película ya existía dentro del listado.

- ordenarPeliculas(): Este método ordena las películas de la misma manera que se hacía en su método homólogo de la clase Eventos y su funcionamiento es el mismo. Devuelve una lista con las películas ordenadas y guarda el resultado en el atributo peliculasOrdenadas.

4.3.4 Película

Esta clase encapsula toda la información que se extrae relativa a una película. Tiene una doble asociación con la clase Cine. Un objeto cine contendrá una referencia a todas las películas que se representen en él. A su vez una película contiene una referencia a todos los cines en donde es representada. Esta doble asociación viene dada por las dos formas distintas que el usuario puede acceder a la información de las sesiones, por cines o por películas respectivamente.

4.3.4.1 Atributos

- String nombre: Este atributo almacena el título de la película.

- String info: En este atributo se almacena la información ofrecida sobre la película: género, duración, versión, etc.

- String genero: En este atributo se almacena el género cinematográfico al que pertenece la película.

- LinkedHashMap<Cine,List<Horario>> horariosCine: En este atributo se almacena un mapa que asocia la lista de horarios de las sesiones disponibles de la película con el cine en dónde se representan.

4.3.4.2 Métodos

- Constructor: Recibe como parámetro las características de la película para asignar valor a sus atributos y se inicializa el mapa de horariosCine.

- getHorariosCine(Cine): Busca en el mapa de horarios cine la lista que corresponde al objeto Cine que se pasa como parámetro. Devuelve la lista de horarios correspondiente.

- getCines(): Devuelve una lista con las referencias a todos los cines en los que se representa la película. Esta lista se obtiene del mapa extrayendo todos los valores de sus claves.

- addCine(Cine): Añade una nueva entrada al mapa de horariosCine utilizando como clave el objeto Cine que se pasa como parámetro e inicializa la lista de horarios a la que está asociado.

- addHorario(Horario, Cine): Añade un nuevo horario a la lista asociada al objeto Cine que se pasa como referencia. Obtiene la lista correspondiente buscando en el mapa de horariosCine por el cine y se añade el horario que se pasa como parámetro.

4.3.5 Horario

Esta clase encapsula la información que se extrae relativa a una sesión concreta de la película.

4.3.5.1 Atributos

- String hora: Este atributo contiene la hora de la sesión en formato de texto.

- String link: Este atributo contiene el enlace a la página web del dominio de entradas.com en el que se puede comprar la entrada correspondiente a esta sesión concreta.

4.3.6 Teatro

Esta clase encapsula la información obtenida sobre un evento de tipo teatro. Extiende de la clase EventoBasico y hereda todos sus atributos y métodos.

4.3.5.1 Atributos

- String link: Este atributo contiene el enlace a la página web del dominio de kulturklik en el que se encuentra la página de información del evento desde la que se puede acceder a la compra de la entrada.

4.4 Paquete de Mapas

Este paquete contiene la actividad Mapa en el que se representan todos los eventos junto con la posición de usuario y permite la interacción del usuario con los marcadores de posición.

4.4.1 Mapa

Esta actividad contiene una vista mapa en la que se representa toda la información obtenida y se pone a disponibilidad del usuario para que interactúe con ella. Esta clase extiende de FragmentActivity [39] que es la actividad que permite incluir una vista de mapa en la interfaz gráfica del usuario.

Esta actividad se ejecuta a petición de usuario cuando pulsa el botón de Ver Eventos en la interfaz principal.

4.4.1.1 Interfaz

La interfaz del mapa se define en el archivo mapa_main.xml. Se compone de una vista del tipo MapView [40] la cual muestra en pantalla un mapa con los datos obtenidos de Google Maps Service y que permite la interactuación del usuario. La vista de mapa

abarca casi la totalidad de la interfaz y se añade un botón en la parte inferior de la pantalla para acceder al menú de opciones.

El menú de opciones contiene tres elementos que permiten diferentes acciones:

- Tipo de mapa: Permite escoger el tipo de representación que se desea para el mapa. Al pulsar sobre esta opción aparece un submenú emergente que permite elegir entre la vista normal, vista satélite o vista híbrida.
- Tipo de transporte: Permite escoger el tipo de transporte que se desea utilizar para calcular los trayectos entre la posición de usuario y los diferentes eventos. Al pulsar sobre esta opción aparece un submenú emergente que permite elegir entre calcular la ruta en coche, andando o en transporte público.
- Ver Detalles ruta: Permite ver un listado con la descripción de los pasos que se han de seguir para recorrer el trayecto calculado. Se mostrarán los detalles de la última ruta activa del último evento seleccionado.

Figura 4.4 Visión de mapa con marcadores

4.4.1.2 Atributos

- GoogleMap mapa: Este atributo almacena el objeto mapa con el que se representan todas las funcionalidades que se pueden hacer sobre la vista MapView de la aplicación.

- HashMap<Marker, EventoBasico> eventMarkerMap: Mapa lógico en el que se asocia un marcador que indica una posición sobre el mapa con el evento al que representa [41].

- EventoBasico eventoActivo: Referencia al elemento que está siendo seleccionado por el usuario. Se utiliza, entre otras cosas, para identificar el destino las rutas cuando se calculan.

- int modoTransporte: Este atributo indica el tipo de transporte que se ha de calcular para las rutas. Se utilizan tres valores a través de constantes: 0, 1 y 2 para identificar a los medios andando, en coche y en transporte público respectivamente.

4.4.1.3 Métodos

- onCreate(Bundle): Este método sobrescribe el de la clase Activity. Se ejecuta cuando se crea la actividad Mapa por primera vez. Recibe como parámetro un objeto de la clase Bundle [42] que permite el tránsito de datos entre actividades.

Primero llama al método sobrescrito de la clase padre y declara el *xml* que se utiliza para el *layout*.

Por motivos del control del flujo de la aplicación, la actividad Mapa puede ser lanzada desde diferentes lugares. En el caso de que sea lanzada desde la actividad CineListaExpandible, se indica que este cine es el que debe aparecer como eventoActivo. Por lo tanto si el objeto Bundle contiene información, asignamos el valor que nos indican al atributo.

Por último crea el escuchador que controla las interacciones con el botón de menú.

- onStart(): Este método sobrescribe el de la clase Activity. Se ejecuta cuando la actividad se hace visible para el usuario. Por lo tanto se ejecuta al crearse y cuando vuelve a la cima de la pila de actividades.

Primero llama al método sobrescrito de la clase padre y se obtiene la referencia al objeto que representa al mapa. Para que el mapa sea visible para versiones de Android anteriores (hasta la 2.2) se utiliza el getSupportFragmentManager() para instanciarlo. Después se obtiene de las preferencias del usuario el tipo de mapa y el tipo de transporte que utiliza por defecto. Por último declara la clase MyInfoWindowAdapter como la que define el formato de las ventanas de información de los marcadores en el mapa.

- onResume(): Este método sobrescribe el de la clase Activity. Se ejecuta cuando la actividad empieza a estar disponible para el usuario. Llama al método colocarMarcadores().

- onCreateOptionsMenu(Menu): Este método sobrescribe el de la clase Activity. Se utiliza para inicializar el contenido del menú de opciones desplegable. Asigna el *layout* del archivo mapa_menu.xml al menú contextual.

- onOptionsItemSelected(MenuItem): Este método sobrescribe el de la clase Activity. Controla la selección de cada una de las opciones disponibles en el menú desplegable de la aplicación. Se implementa mediante un switch - case con todas las opciones del menú, tanto las del menú principal como las de los submenús.

- colocarMarcadores(): Este método se encarga de colocar los marcadores de los eventos y de la posición de usuario sobre el mapa. Rellena el atributo eventMarkerMap, que almacena un mapeo para identificar cada evento con el marcador que le corresponde, y pinta todos los marcadores en el mapa.

En primer lugar, crea un *array* de posiciones en el que se irán añadiendo la posición del usuario y de cada evento. Esto es necesario para poder ajustar la posición del mapa y el zoom para centrar la cámara y mostrar todos los marcadores al mismo tiempo con el mayor nivel de zoom posible.

Después se procede a la creación y colocación de los marcadores. El primero que se coloca será el que indique la posición de usuario que será de color rojo. Luego accede al objeto Eventos, almacenado en la actividad principal, y recorre todas las listas de los diferentes eventos que contiene. En esta primera versión recorrerá primero los eventos de tipo cine y después los de tipo teatro. A los primeros se les asignará marcadores de color azul y a los segundos de color verde.

Cuando se crea un marcador se le indica la posición, un título, un texto y el tipo de icono. El título y el texto aparecerán en la ventana de información cuando el usuario pulse sobre el marcador.

Cuando ya se han asignado todos los marcadores, utilizamos el *array* en el que se han almacenado todas las posiciones y las introducimos en un objeto Builder [43] perteneciente a la clase LatLngBounds [44]. Con este objeto se utiliza el método estático newLatLngBounds() de la clase CameraUpdateFactory [45] que ajusta la posición del mapa y el nivel de zoom para mostrar todos los puntos contenidos en el objeto Builder.

Una vez se han creado y posicionado todos los marcadores, asigna al mapa el escuchador que ha de gestionar la interacción del usuario con los marcadores. El escuchador es común para todo el mapa, así que es necesario identificar el tipo de evento que es para ejecutar una acción u otra:

- Si el marcador pulsado es un cine se muestra una ventana de diálogo emergente que permite al usuario escoger entre tres opciones:
    - Ver la cartelera: Se ejecuta el método mostrarCine(Marker)

    - Cómo llegar: Comprueba si ya existe un evento activo asignado y si es el actual. Si no lo es, comprueba si hay pintadas sobre el mapa rutas pertenecientes al anterior evento y si es así las borra. Se señala el evento actual como eventoActivo y se llama al método calcularCamino() con la posición del marcador.
    - Salir: Cancela el cuadro de diálogo y vuelve al mapa.
- Si el marcador pulsado es un teatro, también se muestra una ventana emergente con las opciones:
    - Ver Página Evento: Se redirige al usuario a la página con la información del evento a partir de la cual se puede acceder a la compra de la entrada.
    - Cómo llegar: La lógica es igual que en caso de los cines.
    - Salir: Cancela el cuadro de diálogo y vuelve al mapa.

- mostrarCine(Marker): Este método lanza la actividad de lista de películas correspondiente al cine al que representa el marcador. Se le incluye un mensaje a la siguiente actividad indicándole desde dónde se le ha llamado para controlar el flujo de la aplicación.

- calcularCamino(LatLng): Este método calcula el trayecto desde la posición de usuario hasta la posición que se le pasa como parámetro.

Primero comprueba si la ruta se ha calculado con anterioridad, si no es así, crea una nueva tarea de la clase TareasDirecciones que se encarga de consultar el servicio web de Google Directions y obtener la ruta entre ambos puntos.

Una vez calculada la ruta se llama al método pintarCaminos().

- pintarCaminos(): Este método se encarga de pintar las rutas que ya estén calculadas del evento activo. Recorre el array de rutas del evento comprobando cuales se han calculado ya y las pinta sobre el mapa. Para pintar en el mapa se utiliza el método addPolyline() de la clase GoogleMap pasándole como parámetro un objeto PolilyneOptions [46] que hemos obtenido a través del servicio web. También guarda una referencia al camino en la posición correspondiente del atributo de la clase EventoBasico, para poder borrar el camino pintado cuando se seleccione otro evento.

Después de pintar el camino, se ajusta la posición de la cámara y el zoom para mostrar el mapa centrado en el camino que se acaba de pintar.

4.4.1.4 Diagrama de clases

Esta actividad contiene un objeto de la clase MyInfoWindowAdapter que maneja las ventanas de información de los marcadores que se crean sobre ella. También utilizará una

referencia al objeto Eventos para tener acceso a toda la información y poder asignar a cada evento un marcador. Esta actividad también puede lanzar la tarea de cálculo de direcciones que obtiene los datos para crear un objeto de tipo Ruta que se asociará con el evento para el que se ha calculado.

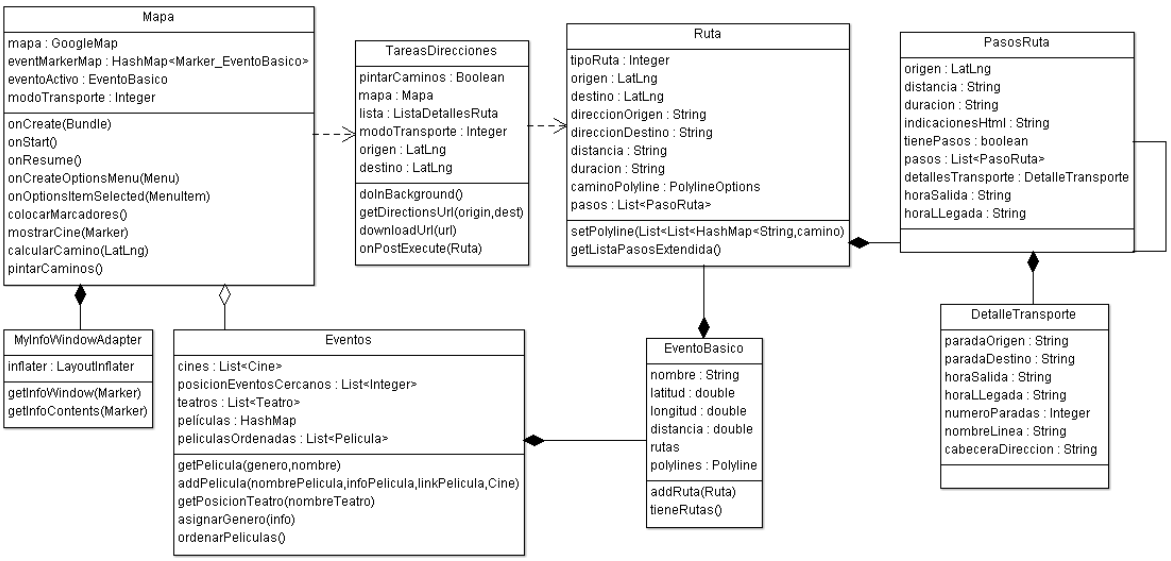

Figura 4.5 Diagrama de clases que utiliza la actividad Mapa

4.4.2 MyInfoWindowAdapter

Esta clase implementa la interfaz InfoWindowAdapter [47] que permite modificar la ventana que aparece asociada a un marcador en el mapa cuando se presiona sobre él. El *layout* que va a presentar es el especificado en el archivo my_info_window.xml. Básicamente en este fichero se define la fuente, tamaño y color del texto del título y de la información de la ventana del marcador. También permite introducir más de una línea de texto en la parte de información, acción que no permite la ventana por defecto.

4.4.2.1 Atributos

- LayoutInflater inflater: Este atributo representa el *layout* sobre el que se va a incluir las modificaciones.

4.4.2.2 Métodos

- getInfoWindow(Marker): Este método es propio de la interfaz InfoWindowAdapter. Lo único que hace es devolver *null*. Esto hará que cuando pulsemos en el marcador pase a ejecutarse el método getInfoContents(Marker) que es el que se modifica.

- getInfoContents(Marker): Este método devuelve una vista que se mostrará cuando el usuario interaccione con un marcador. Crea una vista con el *layout* definido en el archivo my_info_window.xml y asocia el texto relativo al título e información del marcador a los elementos de dicho *layout*.

4.5 Paquete de Lista de Cines

En este paquete se encuentran las actividades que van a mostrar información acerca de los eventos de tipo cine.

4.5.1 CineListaExpandible

Esta actividad muestra una lista con los nombres de las películas que están en su cartelera y los horarios disponibles. Extiende de la actividad ExpandableListActivity [48] que contiene una lista que a su vez contiene otra lista de objetos y controla los eventos que se producen sobre los objetos padres e hijos. En este caso la lista principal será de películas y la lista hija será la de los horarios disponibles para cada película.

4.5.1.1 Interfaz

La interfaz gráfica se define en el archivo cines_lista_expandible.xml y tiene como elemento principal la lista expandible que contendrá la información de películas y horarios. Sobre la parte superior de la lista existe un cuadro de texto que indica el nombre del cine y su información. En la parte inferior se colocarán tres botones:

- Botón + Eventos: Muestra la actividad Mapa con los marcadores correspondientes a todos los eventos.
- Botón Cómo llegar: Muestra la actividad Mapa con los marcadores correspondientes a todos los eventos, calcula la ruta hasta el cine que se está visualizando actualmente y la pinta sobre el mapa.
- Botón Salir: Regresa al usuario a la interfaz principal.

Las entradas de la lista siguen un formato definido en los archivos cine_group_row.xml y cine_child_row.xml para las entradas de películas y horarios respectivamente.

Figura 4.6 Lista de Películas

Figura 4.7 Lista de Películas expandida

4.5.1.2 Atributos

- Cine cine: Es el objeto Cine cuyas características son las representadas en la lista expandible.

- String tipo: Este atributo almacena una cadena de texto que indica la actividad que ha lanzado la actividad actual. Se necesita el parámetro para controlar el flujo de la aplicación.

4.5.1.3 Métodos

- onCreate(Bundle): Este método sobrescribe el de la clase Activity y se ejecuta al crear la actividad.

En primer lugar llama al método de la clase padre e indica el archivo en el que se define el tipo de vista. Después comprueba la información que se le ha pasado desde la actividad que la ha llamado a través del objeto Bundle que recibe como parámetro. Esta información contendrá el tipo de actividad que manda la información y la posición del cine que se quiere mostrar en la lista de cines del objeto Eventos.

Una vez obtenido el cine que se ha de representar se extrae la información para rellenar el cuadro de texto superior de la interfaz con el nombre y la información del cine y se asocia el escuchador de la lista expandible que va a gestionar su comportamiento. El escuchador se crea con el método createExpandableListAdapter().

Además del escuchador de la lista expandible, también crea los escuchadores de los botones definidos en la interfaz.

- createExpandableListAdapter(): Este método crea el objeto SimpleExpandableListAdapter [49] que es el encargado de crear y gestionar los elementos y eventos de la lista expandible. Para crear este objeto se le han de pasar varios objetos a su constructor:

  - contexto: Se debe pasar una referencia a la vista sobre la que se ejecuta este adaptador.
  - Grupo de datos padre: Dentro de este parámetro se pasan al método los datos relativos a la lista de objetos padre. Estos datos se han de formar como una lista de mapas lógicos. Cada elemento de esta lista representa una de las películas que se quieren mostrar y contendrá un mapa con los objetos que se necesiten para crear la lista, utilizando como clave un objeto de tipo String. En este caso los valores que se van a guardar son el título y la información de la película. La lista de películas se obtendrá del atributo listaOrdenada del objeto de la clase Cine.
  - *Layout* del grupo padre: Indica cuál es el formato de una entrada de la lista. En este caso, como se ha dicho antes, el formato de la entrada se especifica en el archivo cine_group_row.xml, que define el tamaño y aspecto del texto perteneciente al título e información de la película.
  - Lista de claves del grupo: Lista de las claves del mapa del grupo de datos padre que contienen la información requerida.
  - Lista de vistas: Lista de las referencias a las vistas incluidas en el layout del grupo padre. Definen la forma en que se representan los objetos que se recuperan a partir de las claves especificadas, es decir, se asocia a cada elemento que se quiere mostrar en la entrada de la lista expandible el formato definido en el *layout*.
  - Datos de los hijos: Los datos relativos a la lista de los objetos de los hijos, se ha de pasar como una lista doble de mapas. Funciona igual que el grupo padre sólo que la información mapeada que se utilizará viene dada por dos coordenadas en vez de una como ocurría en el caso del padre. De esta manera la primera coordenada identifica a la lista como hija de una determinada entrada padre y la segunda coordenada especifica una entrada hija concreta.
  - *Layout* del grupo hijo: Indica cuál es el formato de las entradas que seguirán las entradas hijas. En este caso sólo se mostrará el texto con la hora.
  - Lista de claves del grupo hijo: Lista de las claves del mapeo.
  - Lista de vistas del grupo hijo: Lista de vistas asociadas a los elementos mapeados.

- onChildClick(ExpandableListView, View, groupPosition, childPosition, id): Este método se ejecuta cuando se hace click sobre una entrada hija de la lista expandible, es decir, sobre un horario de una película. Los índices de groupPosition y childPosition indican la película que se ha seleccionado y el horario de sesión respectivamente. Se utiliza el primer índice para buscar en la lista de películas del cine y el segundo para buscar el horario dentro de la lista de la película.

Una vez obtenido el horario seleccionado comprobamos que tiene un link definido. Algunos cines no tienen la posibilidad de comprar entradas a través de entradas.com por lo que puede ocurrir que el campo del enlace aparezca vacío. En este caso se muestra un mensaje emergente informando al usuario de la situación.

Si el enlace sí está disponible aparece un cuadro de diálogo al usuario en el que se le pregunta si quiere comprar o no la entrada para esta sesión. Si se elige Sí, se abre un navegador y se redirige al usuario hasta la página que le permite comprar la entrada; si elige No, se cierra el cuadro de diálogo.

Además de redirigir al usuario a la página para la compra de entrada, se extrae el género de la película, se obtiene el ranking de las preferencias generales de la aplicación y se le suma uno al valor que tenga.

- onKeyDown(keyCode, KeyEvent): Se sobrescribe el método de la clase Activity. Al sobrescribirlo permite alterar el funcionamiento de las teclas del terminal. En este caso se cambia la ejecución del botón "atrás" en función de la actividad que lanzó la actual y así poder volver a ella.

4.5.1.4 Diagrama de clases

Esta actividad contiene un cine del que muestra su cartelera. Desde esta actividad también se puede lanzar el servicio web de cálculo de rutas y asociar un trayecto al objeto Cine.

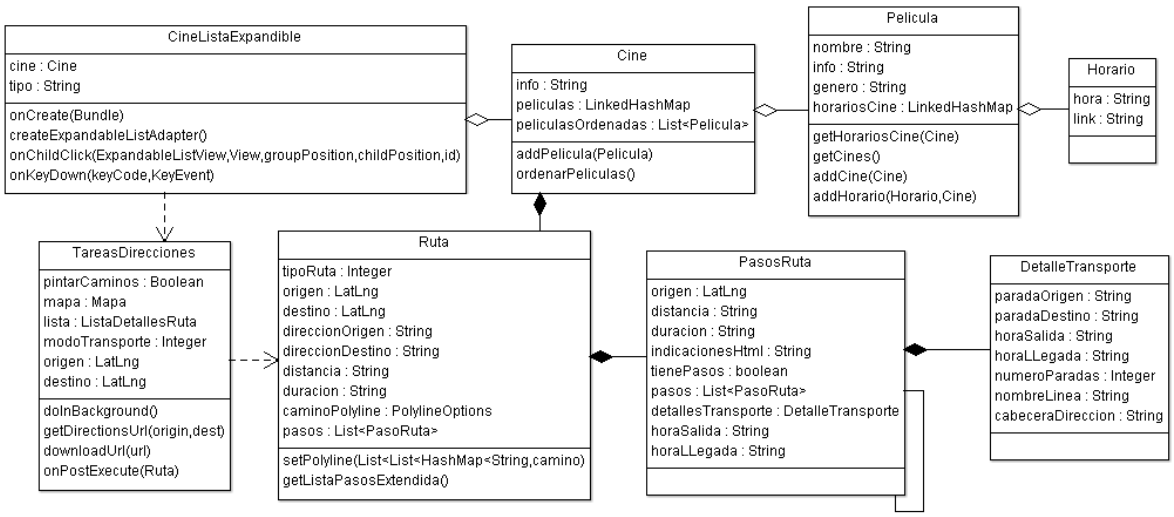

Figura 4.8 Diagrama de clases de la actividad CinesListaExpandible

4.5.2 ListaEventosCercanos

En el caso en el que el usuario se encuentre muy próximo a varios eventos, puede que se añada más de uno a la lista de eventos cercanos. Cuando ocurre esto la aplicación, en vez de mostrar directamente las características de un evento marcado como cercano, como se haría en el caso en el que sólo hubiese uno, se muestra una lista desde la que el usuario puede elegir entre los eventos cercanos para acceder a sus características. Ésta es la clase que representa a esta lista. Extiende de ListActivity [50].

4.5.2.1 Interfaz

La interfaz contendrá como elemento principal una lista simple con los nombres de información de los eventos. El archivo que especifica su apariencia es el cines_cercanos_lista.xml. En la parte superior de la pantalla se mostrará un título en el que se avisa al usuario de que se han encontrado varios eventos muy cerca de su posición, como elemento principal estará la lista con estos eventos, y en la parte de abajo aparecerán dos botones que serán:

- Botón + Eventos: Mostrará la actividad Mapa con todos los eventos ubicados en sus posiciones junto con la del usuario.

- Botón Salir: Elimina esta actividad y devuelve el control a la actividad principal.

Figura 4.9 Ejemplo de lista de eventos cercanos

4.5.2.2 Métodos

- onCreate(Bundle): Este método sobrescribe al de la clase Activity y se ejecuta al crearse la actividad. En primer lugar llama al método de la clase padre e indica el archivo en el que se define el tipo de vista.

Después se utiliza un adaptador de la clase MyArrayAdapter para personalizar los elementos de la lista. Para ello se instancia la vista correspondiente a la lista y se le asigna un objeto de este adaptador.

Una vez que la forma de la lista ya está definida se asigna un escuchador que controle los eventos que se hagan sobre los elementos de la vista. Cuando el usuario haga *click* sobre una entrada de la tabla, se abrirá la actividad CineListaExpandible correspondiente al cine pulsado. Incluimos en la llamada a esta actividad la información que identifica al cine y a la actividad que la está llamando.

4.5.3 MyArrayAdapter()

Esta clase define un adaptador para una lista simple que permite personalizar dicha lista. Extiende de la clase ArrayAdapter [51] que incluye como parámetro de clase una lista de objetos. En este caso será una lista de tipo Evento.

4.5.3.1 Atributos

- Context context: Referencia a la actividad sobre la que se está ejecutando el adaptador.

- ArrayList< Evento > eventos: Lista de objetos de tipo Cine que se utilizarán para definir cada una de las entradas de la lista.

4.5.3.2 Métodos

- Constructor: El constructor admite como parámetro el contexto y la lista de cines como parámetros. Además de almacenar los valores en sus atributos, indica el estilo de cada elemento de la lista definido por el archivo cine_group_row.xml.

- getView(): Este método sobrescribe al de la clase ArrayAdapter<Object>. Es el encargado de crear la vista de cada entrada de la lista. Recibe como parámetro la posición del elemento y la vista del elemento padre y devuelve la vista correspondiente.

Primero con la referencia al contexto de la actividad ListaEventosCercanos se crea un objeto del tipo LayoutInflater que permitirá añadir nuestra entrada personalizada a la vista de la lista. Cada elemento en la lista tendrá dos cuadros de texto, uno para mostrar el nombre del evento y otro para mostrar su información. Asignamos los valores del evento correspondiente buscando en la posición indicada de la lista a los elementos de la nueva vista y se devuelve.

4.6 Paquete de Lista de Películas

En este paquete se incluyen las clases que permiten consultar la cartelera de los cines de los alrededores de la posición del usuario por películas. La funcionalidad principal es mostrar una doble lista expandible que incluya todas las películas disponibles, dentro de ellas los cines en los que se reproducen y dentro de cada cine los horarios de sesión correspondientes. Las películas vendrán ordenadas según las preferencias del usuario y su importancia.

4.6.1 PeliculasListaExpandible

Esta es la actividad principal que incluye la doble lista expandible. Extiende de la clase ExpandableListActivity que representa una lista expandible. En este caso se utiliza la clase PeliculasAdaptadorNivel1 como adaptador de la lista expandible para personalizarla y conseguir una lista con dos niveles. En la lista principal se enumeran las películas ordenadas, en los elementos hijos de primer nivel se listan los cines en los que se proyecta esa película y en los hijos de segundo nivel se muestran los horarios de sesión en ese cine.

4.6.1.1 Interfaz

El diseño de la interfaz está definido en el fichero peliculas_lista_expandible.xml. Se compone de un cuadro de texto como título, la lista expandible como elemento principal y un botón en la parte inferior para devolver al usuario al menú principal.

Cada nivel de la lista utiliza un archivo *xml* para definir su diseño:

- Nivel Padre: En cada elemento de este nivel se muestra el título e información de la película a la que representa. Además, en el extremo derecho, se incluye un botón que permite al usuario ver en un mapa los cines que proyectan dicha película. Se define en el archivo peliculas_group_row.xml.

- Nivel Hijo de Nivel 1: En cada elemento de este nivel se muestra el nombre del cine en el que se representa la película asociada y la información del cine. Se define en el archivo peliculas_child_row1.xml.

- Nivel Hijo de Nivel 2: En cada elemento de este nivel se muestra el horario de la sesión a la que se representa la película asociada. Se define en el archivo peliculas_child_row2.xml. Al pulsar sobre este elemento se accederá a la página web que permite comprar la entrada.

Figura 4.10 Lista expandible de películas

4.6.1.2 Métodos

- onCreate(Bundle): Este método sobrescribe el de la clase Activity que se ejecuta cuando se crea la actividad. Primero llama al método de la clase padre e identifica el archivo *xml* que define la interfaz. Después crea una instancia de la clase PeliculasAdaptadorNivel1 y lo asigna como controlador de la lista. Por último asigna el escuchador del botón para volver al menú principal.

4.6.1.3 Diagrama de clases

Esta actividad contiene un objeto vista para definir la lista expandible. Este objeto vista utiliza un adaptador de primer nivel para manejar los eventos sobre él. Este adaptador a su vez contiene al adaptador de segundo nivel para manejar los eventos que se producen sobre el nivel dos y tres de la lista. El adaptador de nivel uno extrae información de películas y cines y el de nivel dos de cines y horarios para crear los elementos de la lista.

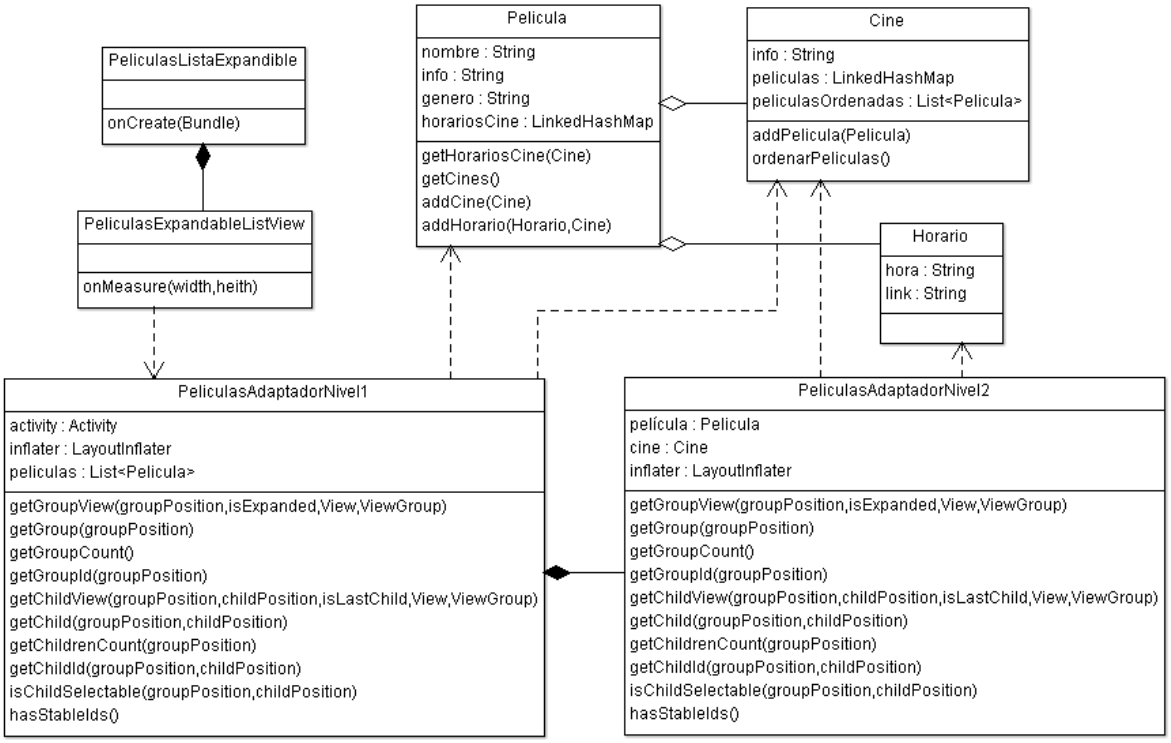

Figura 4.11 Diagrama de clases de la actividad PeliculasListaExpandible

4.6.2 PeliculasExpandableListView

Esta clase extiende de la clase ExpandableListView que es la vista que define una lista expandible. Necesitamos esta clase para sobrescribir su método onMeasure() que se encarga de controlar el espacio que ocupan los nodos hijos para pintarlos en pantalla. En este caso no sirve el valor por defecto ya que los elementos hijos tienen a su vez otra lista y por lo tanto se requiere mayor tamaño de pantalla para mostrarlos.

4.6.2.1 Métodos

- onMeasure(width,heith): Sobrescribe el método de la clase padre. Para conseguir que la doble lista se dimensione correctamente se utiliza el método estático makeMeasureSpec de la clase MeasureSpec [52] que recibe como parámetros un valor de tamaño y un modificador para indicar que ese tamaño es el exacto, el máximo o el mínimo. De esta manera se escoge un valor alto de tamaño con el modificador de máximo y así se ajustará el tamaño a los elementos mostrados.

4.6.3 PeliculasAdaptadorNivel1

Esta clase extiende de la clase BaseExpandableListAdapter. Se utiliza como adaptador de la lista PeliculasListaExpandible. Aporta la funcionalidad para crear la doble lista expandible y controlar los eventos que se producen en ella.

4.6.3.1 Atributos

- Activity activity: Referencia a la actividad en la que se encuentra la lista expandible que ha de manejar.

- LayoutInflater inflater: Objeto que permite la personalización de las vistas de la interfaz.

- List<Pelicula> peliculas: Lista con las películas ordenadas por las preferencias del usuario y por importancia. Cada película contiene referencias a los objetos Cine y Horario necesarios para crear la lista doble deseada.

4.6.3.2 Métodos

- Constructor: Este constructor recibe como parámetro la actividad que contiene a la lista de la cual va a funcionar como adaptador. Del contexto de esta actividad extrae el objeto LayoutInflater que permite personalizar las vistas del interfaz y se asigna la lista de películas ordenadas de la clase principal Kinetea al atributo peliculas.

- getGroupView(groupPosition, isExpanded, View, ViewGroup): Este método sobrescribe el de la clase padre. Devuelve la vista para cada elemento de la lista principal. Recibe como parámetros la posición en el grupo del elemento del que estamos devolviendo la vista; si el elemento está expandido o no; la vista del objeto en un instante anterior, por si ya se tiene en memoria la vista para este elemento no volver a crearlo, y la vista del elemento padre al que está ligado el elemento.

Lo primero que se hace es comprobar si la vista del elemento ya se encuentra creada en memoria, si es así se recupera y se devuelve, si no, hay que crearla. Si se ha de crear la vista, se utiliza el objeto LayoutInflater para asignar a la vista el diseño que se ha definido en el archivo peliculas_group_row.xml. Con la posición del elemento obtenemos la película que corresponde buscando en la lista de películas ordenadas el índice marcado por el parámetro groupPosition, y se asignan los valores correspondientes de título e información a la vista del elemento.

Después, se asigna un escuchador para el botón incluido en el elemento que lanzará la actividad PeliculasMapa, en la que se muestran los cines en los que se representa esta película. Al lanzar la actividad se incluye un Bundle con la información del género y

nombre de película que hará que pueda recuperarla del mapa de películas de la actividad principal Kinetea.

- getGroup(groupPosition): Este método sobrescribe el de la clase padre. Devuelve el objeto asociado a cada posición. En este caso devuelve la película en la posición indicada en el *array*.

- getGroupCount(): Este método sobrescribe el de la clase padre. Devuelve el número de elementos existente en el grupo. Devuelve el número de películas incluidas en el array de películas ordenadas.

- getGroupId(groupPosition): Sobrescribe el método de la clase padre. Este método se encarga de asignar un identificador distinto para cada elemento del grupo. En este caso se utiliza la misma posición del grupo como identificador.

- getChildView(groupPosition, childPosition, isLastChild, View, ViewGroup): Es un método análogo a getGroupView() para los elementos de la lista hija de primer nivel. En esta ocasión la vista será otra lista expandible de la clase PeliculasExpandableListView, comprueba si ya existe y si no se crea.

A partir de los métodos getGroup() y getChild() obtenemos los objetos película y cine asociados a los elementos padre e hijo de primer nivel respectivamente. Estos objetos se pasan como parámetro, junto con la actividad de la que dependen las vistas, al constructor de la clase PeliculasAdaptadorNivel2. Después se añade este adaptador como controlador de los eventos de la vista de la segunda lista expandible.

Por último se establece un escuchador para controlar los eventos que se producen al pulsar sobre los elementos hijos de segundo nivel que pertenecen a la segunda lista expandible. El escuchador lanzará el método onChildClick() que obtiene el horario de sesión asociado con el elemento pulsado y muestra un cuadro de diálogo emergente con la opción de comprar la entrada o no. Si se desea comprar la entrada se abre un explorador y se redirige al usuario a la página correspondiente para realizar la compra, si no, se cierra el cuadro de diálogo.

- getChild(groupPosition, childPosition): Este método sobrescribe el de la clase padre. Devuelve el objeto asociado al elemento hijo. En este caso devuelve el cine correspondiente a la posición childPosition en la lista de cines que contiene el objeto situado en la posición groupPosition de la lista de películas ordenadas.

- getChildrenCount(groupPosition): Este método sobrescribe el de la clase padre. Devuelve el número de elementos hijo existente en un grupo. En este caso devuelve el número de cines en los que se proyecta la película situada en la posición indicada.

- getChildId(groupPosition, childPosition): Sobrescribe el método de la clase padre. Este método se encarga de asignar un identificador distinto para cada elemento del grupo. En este caso se utiliza como identificador la posición del grupo más un número lo suficientemente alto para evitar conflictos.

- isChildSelectable(int groupPosition, int childPosition): Sobrescribe el método de la clase padre. Devuelve un booleano que indica si el elemento hijo definido por las coordenadas dadas es seleccionable o no. En este caso todos los elementos lo son, por lo que se devuelve siempre verdadero.

- hasStableIds(): Sobrescribe el método de la clase padre. Devuelve un booleano que indica si los identificadores asociados a los elementos tanto padres como hijos son estables en el tiempo o si en cambio pueden variar. En este caso todos los elementos tienen un identificador fijo, por lo que se devuelve siempre verdadero.

4.6.4 PeliculasAdaptadorNivel2

Esta clase extiende de la clase BaseExpandableListAdapter. Se utiliza como adaptador de la lista de segundo nivel en la doble lista expandible de películas. Controla los eventos que se realicen sobre ella.

Se debe recordar que en el método getChildView() del adaptador PeliculasAdaptadorNivel1 que utilizábamos para manejar la lista de primer nivel, se genera una vista de lista expandible para cada hijo de nivel 1 que corresponde a los cines. Como se crea esta vista para cada hijo, lo que tendremos será una lista expandible con un solo elemento padre, el cine, y varios elementos hijos, los horarios.

4.6.4.1 Atributos

- Pelicula película: Película sobre la que tenemos la información

- Cine cine: Cine del que se obtienen los horarios de las sesiones disponibles para la película.

- LayoutInflater inflater: Objeto que permite la personalización de las vistas de la interfaz.

4.6.4.2 Métodos

- Constructor: Este constructor recibe como parámetro la actividad que contiene a la lista de la cual va a funcionar como adaptador y la película y cine sobre los que estamos mostrando la información de los horarios de las sesiones. También extrae el objeto LayoutInflater que permite personalizar las vistas del interfaz.

- getGroupView(groupPosition, isExpanded, View, ViewGroup): Este método sobrescribe el de la clase padre. Devuelve la vista para cada elemento de la lista principal de la lista que se crea en el segundo nivel. En este caso se referiría a la posición del cine. Es análogo al método descrito para el adaptador que se utiliza para el adaptador de nivel 1.

En este caso se obtiene el diseño de la vista del archivo peliculas_child_row1.xml y se asigna al elemento el nombre y la información relativas al objeto cine.

- getChildView(groupPosition, childPosition, isLastChild, View, ViewGroup): Este método sobrescribe al de la clase padre. Devuelve la vista del elemento hijo. También es análogo al que se ha descrito en el adaptador de nivel 1 aunque más sencillo, ya que el escuchador que controla sus eventos ya se ha implementado en el nivel superior.

Obtiene el diseño de vista del esquema peliculas_child_row2.xml y asigna a cada elemento hijo el horario al que representa.

A parte de estos métodos también se implementan los demás métodos de BaseExpandableListAdapter adaptador de nivel 1 que, por su sencillez y similitud a éstos, no se van a detallar de nuevo.

4.6.5 PeliculasMapa

Esta actividad muestra un mapa con marcadores que indican la posición de los cines en los que se proyecta una determinada película y la del usuario. Extiende de la clase Mapa, que se ha descrito anteriormente, y con la que comparte gran parte de su funcionalidad.

4.6.5.1 Interfaz

El diseño del interfaz está definido en el archivo peliculas_map.xml. El elemento principal es un mapa en el que se muestran los marcadores correspondientes a los cines y al usuario. Debajo del mapa se muestran tres botones:

- Botón Menú Principal: Para volver al menú principal.

- Botón de Opciones: Abre el menú de opciones. El menú es el mismo que se utiliza en la clase padre.
- Botón de Volver a Horarios: Permite volver a la actividad anterior.

Figura 4.12 Mapa de cines en los que se representa la película escogida

4.6.5.2 Atributos

- Pelicula pelicula: Película acerca de la cual se están mostrando los cines en los que se proyecta.

- GoogleMap mapa: Elemento mapa.

4.6.5.3 Métodos

- onCreate(Bundle): Este método sobrescribe el de la clase Activity. En primer lugar llama al método de la clase padre e identifica el esquema que define el diseño de la actividad. Después, obtiene de la actividad que ha llamado a ésta el género y el nombre de la película, que se utiliza para buscar el objeto en el mapa de películas de la actividad principal Kinetea. Por último asigna escuchadores para cada uno de los botones del interfaz.

- onStart(): Este método sobrescribe el de la clase Activity. Es análogo al de la clase Mapa, la única diferencia está en el identificador de la vista que se asocia al mapa. En este caso la vista es map_peliculas en lugar de mapa.

- onResume(): Este método sobrescribe el de la clase Activity. Llama al método colocarMarcadores().

- onKeyDown(): Se sobrescribe el método de la clase Activity. Al sobrescribirlo permite alterar el funcionamiento de las teclas del terminal. En este caso se cambia la ejecución del botón Atrás para volver a la actividad de lista de películas.

- colocarMarcadores(): Este método es análogo al de la clase padre Mapa. La diferencia entre ambos reside en que en este caso sólo se crean marcadores para los cines en los que se representa la película y en el comportamiento del escuchador que maneja los eventos sobre los marcadores.

En este caso, al pulsar sobre el marcador, aparecerá la ventana de diálogo emergente con la opción de calcular el camino hasta la posición o de cerrar la ventana.

- pintarCaminos(): Este método es análogo al de la clase padre Mapa. La única diferencia es la referencia que se hace al identificador de vista del mapa.

4.6.5.4 Diagrama de clases

Esta actividad contiene la información de los cines que están relacionados con una determinada película. Desde esta actividad es posible lanzar el cálculo de rutas y almacenarlos en el objeto cine correspondiente.

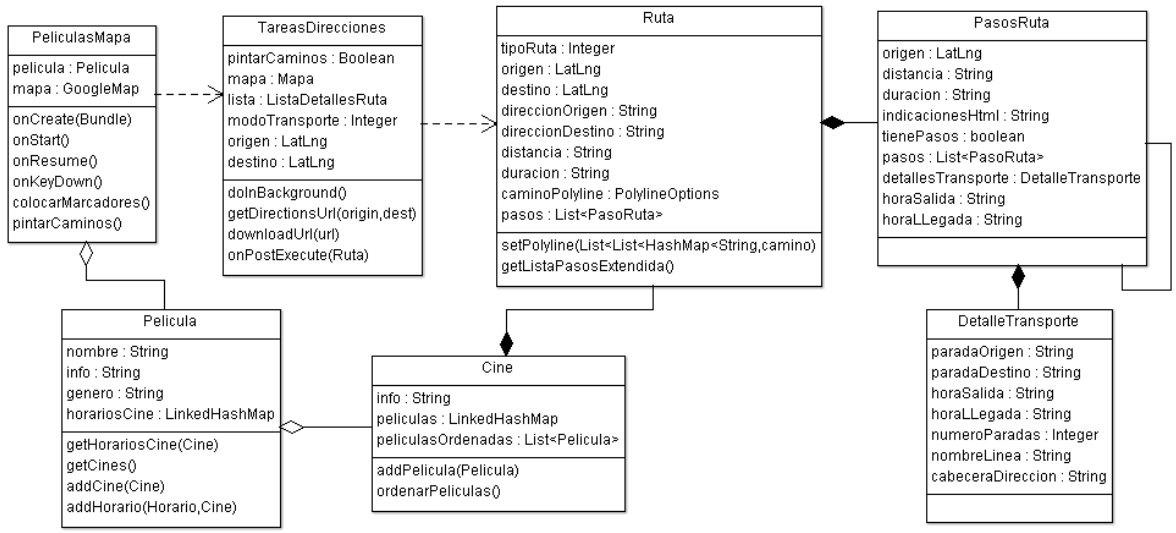

Figura 4.13 Diagrama de clases de la actividad PeliculasMapa

4.7 Paquete de **rutas**

En este paquete se van a incluir todas las clases que se necesitan para ofrecer el servicio de cálculo de rutas. Se divide en dos subpaquetes, uno con los elementos pasivos, que son aquellas clases que nos permiten almacenar la información relativa a las rutas en objetos, y otra con las clases activas, que permiten la descarga, parseo y visualización de las rutas.

Se recuerda que el cálculo de rutas se hará mediante una petición al servicio web de Google Directions.

4.7.1 Ruta

Esta clase encapsula todas las características generales de un trayecto entre dos puntos.

4.7.1.1 Atributos

- int tipoRuta: Tipo de la ruta que indica para qué modo de transporte se ha calculado la ruta.

- LatLng origen: Indica las coordenadas de la ubicación origen.

- LatLng destino: Indica las coordenadas de la ubicación destino.

- String direccionOrigen: Descripción como texto de la dirección que corresponde a las coordenadas de origen.

- String direccionDestino: Descripción como texto de la dirección que corresponde a las coordenadas de destino.

- String distancia: Distancia que existe entre el origen y el destino indicado como una cadena de texto.

- String duracion: Duración aproximada del trayecto en condiciones normales y en el modo de transporte especificado.

- PolylineOptions caminoPolyline: Objeto que contiene toda la información para pintar sobre el mapa un Polyline que corresponda con la ruta.

- List<PasoRuta> pasos: Una lista con los diferentes pasos que componen la ruta

4.7.1.2 Métodos

- setPolyline(List<List<HashMap<String, String>>> camino): Este método se encarga de asignarle valor al atributo caminoPolilyne creándolo a partir del formato de información obtenido del servicio web.

La primera lista representa cada una de las diferentes opciones que hay para recorrer la ruta, en nuestro caso sólo contemplamos la que ofrece el servicio como óptima. La segunda lista representa a cada uno de los pasos que tiene la ruta. Por último, el mapa contiene los valores de las coordenadas de cada punto de la ruta.

Se extraen todos los puntos y se añaden al objeto Polyline, también se le establece el grosor de la línea marcada en el mapa y se asigna un color para cada tipo de ruta.

- getListaPasosExtendida(): Este método crea una lista ampliada de todos los pasos que componen una ruta. Esta lista está destinada a servir de fuente de información a la actividad ListaDetallesRuta para mostrar los detalles de cada ruta paso por paso.

Se diferencian dos tipos de acciones dependiendo de si la ruta ha sido calculada para los medios de transporte a pie y en coche o que haya sido calculada para el transporte público:

En el caso en el que la ruta haya sido calculada para el medio de transporte a pie o en coche, se devuelve la lista de pasos añadiendo un elemento al inicio de la lista con la información del origen de la ruta y otra al final con la información del destino de la ruta.

En el caso en el que la ruta haya sido calculada para el medio de transporte público, además de añadirse los pasos descriptivos de origen y destino, se añaden los subpasos que puedan tener cada uno de sus pasos de ruta.

4.7.2 PasosRuta

Esta clase encapsula las características de cada uno de los pasos de la ruta que se obtiene del servicio web. Se pueden tener dos tipos de pasos de ruta dependiendo del medio de transporte para el que se esté calculando la ruta. Para el transporte público se añaden nuevos atributos como la hora de salida y llegada, y cada paso puede contener a su vez subpasos.

4.7.2.1 Atributos

- LatLng origen: Coordenadas del punto de origen.

- String distancia: Distancia recorrida en este paso de la ruta.

- String duracion: Duración aproximada del trayecto en condiciones normales y en el modo de transporte especificado para este paso de la ruta.

- String indicacionesHtml: Texto de las indicaciones que debe seguir el usuario para realizar el trayecto por la ruta especificada.

- boolean tienePasos: Booleano que indica si este paso de la ruta contiene subpasos. Esto se da sólo para pasos del modo de transporte público.

- List<PasoRuta> pasos: Lista con los subpasos que contiene este paso. Esto se da sólo para pasos del modo de transporte público.

- DetalleTransporte detallesTransporte: Detalles relativos al medio de transporte que en el que se realiza este paso. Esto se da sólo para pasos del modo de transporte público ya que el trayecto puede ser en tranvía, metro, autobús, etc.

- String horaSalida: Sólo válido para pasos de transporte público. Indica la hora de salida del transporte público desde el origen.

- String horaLLegada: Sólo válido para pasos de transporte público. Indica la hora de llegada del transporte público al origen.

4.7.2.2 Métodos

- Constructores: Tiene varios constructores en el que varían el número de parámetros de entrada, ya que el número de atributos que instanciaremos no será el mismo dependiendo de si se trata de un paso de transporte público u otro transporte, o de si se corresponde a un paso o a un subpaso.

4.7.3 DetalleTransporte

Cuando se calcula una ruta con el medio de transporte público existen pasos en los que se han de utilizar diferentes vehículos tales como el metro, el autobús, etc. Esta clase encapsula las características de los transportes para cada uno de los pasos.

4.7.3.1 Atributos

- String paradaOrigen: Nombre la parada origen del recorrido.

- String paradaDestino: Nombre de la parada destino del recorrido

- String horaSalida: Hora de salida desde el origen en formato de texto.

- String horaLLegada: Hora de llegada al destino en formato de texto.

- int numeroParadas: Número de paradas existentes entre el origen y el destino.

- String nombreLinea: Nombre de la línea del transporte que se está utilizando.

- String cabeceraDireccion: Nombre de la estación final de la línea en la dirección que se está utilizando.

4.7.4 TareasDirecciones

Esta clase extiende de AsyncTask y se encarga de hacer la conexión con el servicio web de Google Directions para obtener la información de la ruta en un hilo de ejecución paralelo al hilo principal.

Esta tarea se puede ejecutar desde dos lugares diferentes:

- Desde la lista de detalles de ruta en la que se quiere ver la lista de detalles de otro medio de transporte al que se está viendo actualmente y que aún no ha sido calculado. En este caso se mostrará la lista de detalles del nuevo medio de transporte.
- Desde el mapa ante la petición de cómo llegar desde un determinado evento. En este caso se pintará la ruta sobre el mapa.

4.7.4.1 Atributos

- boolean pintarCaminos: Indica si se ha llamado a la tarea desde el mapa o no y, por lo tanto, se ha de pintar el trayecto sobre éste.

- Mapa mapa: Referencia al mapa desde el que se llama a la tarea.

- ListaDetallesRuta lista: Referencia a la lista de detalles de trayecto desde la cual se llama a la tarea.

- int modoTransporte: Modo transporte para el que se ha de calcular la ruta.

- LatLng origen, destino: Coordenadas de los puntos origen y destino.

4.7.4.2 Métodos

- Constructores: La clase tiene dos constructores diferentes, uno para cada una de las actividades desde donde puede ser lanzada la tarea.

En ambos casos se pasan como parámetros las coordenadas de origen y destino en objetos de la clase LatLng. También tiene como parámetro de entrada una referencia a la actividad que crea el objeto, un objeto Mapa en un caso y un objeto de ListaDetallesRuta en el otro.

En el constructor, aparte de asignar los valores que se ha visto que se pasan como parámetros a los atributos correspondientes, también se asigna el valor del modo de transporte. En el caso de que se haya llamado desde la actividad Mapa, el medio de transporte se obtiene del objeto que referencia a esta clase. En el otro caso se pasa como parámetro del constructor.

- doInBackground(): Este método sobrescribe el de la clase AsyncTask. Contiene la mayor carga computacional del hilo secundario. Primero se utiliza el método getDirectionsUrl() para obtener la dirección http con la que se llama al servicio web. Después con el método downloadUrl() se conecta a la red y ejecuta la petición. Una vez que ya se ha obtenido la respuesta se le pasa al parseador para que cree los objetos a partir del texto plano *json*.

- getDirectionsUrl(origin, dest): Este método recibe como parámetros los objetos LatLng con las coordenadas del punto de origen y de destino del trayecto. Con esta información y el medio de transporte almacenado en los atributos de la clase, se crea la dirección http que hará la petición al servicio web. Si el tipo de ruta es de transporte público, se ha de añadir además el horario de inicio de la ruta, para el cual se tomará la fecha y hora actual.

- downloadUrl(url): Este método crea una conexión a internet y realiza la petición al servicio web a partir de la *url* calculada con el método anterior y que recibe como parámetro. Recibe y lee la respuesta en formato *json* y lo almacena en una cadena de texto plano que devuelve.

- onPostExecute(Ruta): Este método sobrescribe el de la clase AsyncTask. Se ejecuta cuando termina el método doInBackground() y recibe como parámetro el objeto de la clase Ruta creado en este método. Dependiendo de la actividad que ha llamado a la tarea pinta el camino calculado sobre el mapa de la actividad Mapa o muestra la lista con los detalles de la ruta por pasos en la actividad de ListaDetallesRuta.

4.7.5 DireccionesJSONParser

Esta clase actúa como parseador de la respuesta *json* obtenida del servicio web de Google Directions.

4.7.5.1 Métodos

- parse(JSONObject, tipoRuta): Este es el método de la clase que cumplirá su funcionalidad principal. Recibe como parámetro el objeto *json* obtenido como respuesta del servicio web y el tipo de ruta que es, es decir, el medio de transporte para el que se ha calculado la ruta, ya que el esquema de la respuesta varía dependiendo de este factor. Devuelve un objeto Ruta con las características de la respuesta del servicio web.

La respuesta, como se ha visto en el capítulo 2.6.4, contiene un *array* de rutas posibles de las cuales se escoge la primera, que corresponde a la ruta óptima considerada por el servicio web. De esta ruta se extraen todos los datos y se encapsulan en un objeto de la clase Ruta. Después se recorren todos los pasos que se especifican y se encapsulan en objetos PasoRuta que irán asociados al objeto Ruta creado.

La información que nos permite extraer los puntos por los que pasa la ruta y así poder crear los segmentos que se pintan en el mapa, viene codificada con un formato específico basado en bits en lugar de en caracteres. Se utiliza el método decodePolylines() para extraer las coordenadas de todos los puntos y poder crear así la línea que representará a la ruta en el mapa.

Si el tipo de ruta es de transporte público, se han de considerar más aspectos. Primero se comprueba si este paso de la ruta contiene subpasos, si es así, se recorren éstos y se almacenan en la lista de pasos del objeto PasoRuta del que dependen. Si no contiene subpasos se especifica los detalles del transporte público correspondiente a ese paso y se almacenan los datos en un objeto DetalleTransporte.

Una vez que se ha recorrido todo el array y almacenado toda la información en el objeto Ruta y en los objetos TipoRuta y DetalleTransporte asociados a éste, se devuelve dicho objeto.

- decodePoly(String encoded): Este método se encarga de procesar el texto con formato binario en una lista de coordenadas LatLng que representan a los puntos por los que pasa la ruta.

4.7.6 ListaDetallesRuta

Esta actividad muestra una lista con las indicaciones de todos los pasos a seguir para recorrer la ruta calculada entre la posición de usuario y el evento seleccionado. Extiende de ListActivity.

4.7.6.1 Interfaz

Tiene como elemento principal una lista simple en la que se mostrará el origen y destino de la ruta con todos los pasos intermedios que se han de seguir para llegar desde uno hasta el otro. Los elementos que representan al origen y destino se diferencian del resto elementos. El esquema que sigue la actividad es el definido en el archivo detalle_rutas_lista.xml.

En la parte superior de la actividad se muestra como título el nombre del lugar en el que se produce el evento para el que se ha calculado la ruta. Debajo de éste se introducen tres botones, uno para cada medio de transporte. El diseño de estos tres botones se realiza para que den la apariencia de pestañas.

En la parte inferior de la actividad aparece un botón para volver a la actividad anterior.

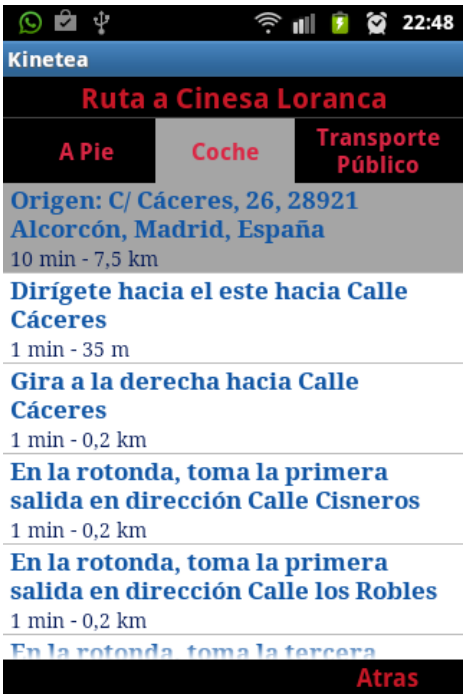

Figura 4.14 Lista de detalles de ruta

4.7.6.2 Atributos

- EventoBasico elemento: Referencia al evento para el cuál se ha calculado la ruta.

- int modoTransporte: Modo de transporte para el cuál se está modificando la ruta.

- ListView listaDetallesRuta: Objeto vista que representa la lista de la actividad en el que mostraremos la información de la ruta.

4.7.6.3 Métodos

- onCreate(Bundle): Este método sobrescribe al de la clase Activity. Se ejecuta al crearse la actividad por primera vez. En primer lugar llama al método de la clase madre e instancia el esquema que define su interfaz.

Después se extraen del objeto Bundle que ha sido enviado por la actividad anterior, la información del medio de transporte y del tipo de evento sobre los que se está representando los detalles de ruta. El tipo de evento es necesario para saber a qué lista de la clase principal Kinetea hay que recurrir para obtener el evento al que corresponde la ruta.

A continuación se llama al método mostrarDetallesRuta().

Por último se crean los escuchadores para implementar las acciones de los botones del interfaz. Los tres botones que indican los tres medios de transporte funcionan de la misma manera, primero comprueban si la ruta ya está calculada, si es así, llaman al método mostrarDetallesRuta() para mostrar los detalles, si no lo está, se llama al método crearDialogoAlerta(). En ambos casos se modifica el valor del atributo modoTransporte.

- mostrarDetallesRuta(): Este método se encarga de obtener la ruta de la que se desea mostrar sus detalles y crear la lista correspondiente. Para obtener la ruta comprueba el elemento correspondiente al modo de transporte activo del *array* de rutas del atributo evento. A partir de esta ruta obtenemos la lista de pasos extendida y se la pasamos como parámetro al constructor de la clase privada MyArrayAdapter que instanciará un adaptador para manejar la vista de la lista.

Por último se llama al método modoTransporteActivado() para actualizar la interfaz.

- modoTransporteActivado(modoTransporte): Este método recibe como parámetro el modo de transporte activado en el momento y cambia el diseño de los botones según el modo seleccionado para dar la apariencia de pestañas.

- crearDialogoAlerta (): Si se selecciona uno de los botones que representan a los modos de transporte y aún no se ha calculado la ruta para ese medio, se muestra una ventana de diálogo al usuario en la que se le informa de que la ruta no está aún disponible y se le ofrece la opción de calcularla. Si el usuario decide calcular la ruta se ejecuta la tarea de TareasDirecciones.

4.7.7 MyArrayAdapter

Esta clase es una clase privada dentro de la clase ListaDetallesRuta que extiende de la clase ArrayAdapter a la que se le pasa como parámetro un objeto de la PasoRuta. Funciona como adaptador de la lista y maneja su aspecto.

4.7.7.1 Atributos

- Context context: Referencia a la actividad que contiene a la lista.

- List<PasoRuta> pasosRutas: Lista extendida con todos los pasos y subpasos de la ruta que se detalla más los elementos descriptivos de origen y destino.

4.7.7.2 Métodos

- getView(position, View, ViewGroup): Este método sobrescribe al de la clase padre. Recibe como parámetros la posición del elemento a mostrar, la vista de ese elemento si ya se ha creado con anterioridad y la vista de la lista padre que contiene a estos elementos. Devuelve la vista para el elemento en la posición indicada.

Primero se crea un objeto LayoutInflater a partir del contexto de la actividad que contiene a la lista para poder personalizar la vista de ésta.

Para los pasos inicial y final que corresponden al origen y destino de la ruta los diferenciamos del resto cambiando el color del fondo de las entradas correspondientes.

Para todos los pasos se muestra las indicaciones acompañado con la duración y distancia correspondiente al paso de la ruta. Si el medio es transporte público se tiene en cuenta si el paso contiene subpasos o si tiene detalles de transporte para crear el texto que se introducirá en indicaciones.

Capítulo 5. Pruebas

En este capítulo se presentan las pruebas de integración y funcionalidad que se han realizado sobre la aplicación.

En primer lugar se presentarán las pruebas sobre el funcionamiento básico en la que se comprobará la navegación de la aplicación, el correcto funcionamiento de las rutinas de obtención de datos y la presentación de los datos al usuario. Para este tipo de pruebas se ha instalado la aplicación en el terminal de varios usuarios y se ha comprobado la respuesta de ésta para cada uno de los diferentes tipos de prueba.

En segundo lugar se simulará la posición del usuario para conseguir disparar diferentes funcionalidades en la aplicación. En esta ocasión se ha utilizado un único móvil de pruebas que se ha conectado en modo depuración al entorno de Eclipse. Se ha identificado las zonas que cumplen con las características en las que se desea probar la aplicación y con ayuda de Google Earth se ha obtenido sus coordenadas. Estas coordenadas se introducen en el simulador de posición del emulador de Android y se envían al dispositivo, de esta manera se ejecuta el método de actualización de posición en el dispositivo. Para realizar esta acción se utiliza la perspectiva DDMS de Eclipse, que muestra una ventana como la de la figura 5.1, en la que se señala la pestaña Emulator Control y se introducen los valores de las coordenadas en la sección de Location Controls. Después, para enviar las coordenadas al dispositivo se pulsa el botón Send [53].

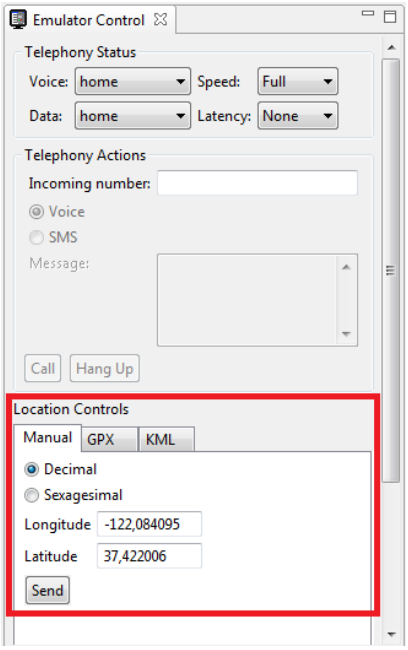

Figura 5.1 Pantalla del emulador de posición

5.1 Pruebas generales

En este apartado se van a describir las pruebas que se han hecho para el funcionamiento más general de la aplicación. Las pruebas las vamos a agrupar por la actividad en las que se ejecutan las acciones que vamos a verificar.

- Kinetea:

Prueba	Resultado	Observaciones
Comprobación de GPS activo.	Ok	Al iniciarse la aplicación se comprueba si el GPS está activo en ese momento o no.
Opción de habilitar GPS si no está habilitado.	Ok	Si el GPS no se encuentra activo se recomienda al usuario la activación del mismo a través de un cuadro de diálogo. Si el usuario decide activarlo se le redirige a la opción correspondiente en su teléfono.
Elección del mejor sistema de localización.	Ok	Al iniciarse la aplicación se comprueba los sistemas de localización disponibles y se utiliza el que en ese momento esté ofreciendo mejores prestaciones.
Geolocalización del usuario.	Ok	La aplicación es capaz de posicionar

		correctamente al usuario y guardar sus coordenadas.
Obtención de eventos de tipo cine.	Ok*	La aplicación obtiene correctamente todos los eventos de tipo cine que contiene la página web de www.google.es/movies.
Menú de opciones.	Ok	El menú de opciones se despliega correctamente al pulsar el botón de opciones o el botón de menú del móvil (si lo tiene).
Lanzar actividad de información.	Ok	Al pulsar el botón de Info se lanza la actividad que muestra la información acerca de la aplicación y desde ella se vuelve a la actividad principal.
Lanzar actividad de ayuda.	Ok	Al pulsar el botón de Ayuda se lanza la actividad que muestra la ayuda para ejecutar la aplicación y desde ella se vuelve a la actividad principal.
Cambiar opciones de búsqueda.	Ok	Se accede al submenú de opciones de búsqueda y se permite elegir entre obtener información de eventos de tipo cine, teatro o ambos.
Cambio de tipo de búsqueda persistente entre una sesión y otra.	Ok	La elección del tipo de búsqueda se almacena en las preferencias generales de la aplicación y se mantiene de una sesión a otra.
Lanzar actividad de mapa.	Ok	Se lanza la actividad que contiene al mapa al pulsar el botón Ver Mapa.
Lanzar actividad de lista películas.	Ok	Se lanza la actividad de lista de películas al pulsar el botón Lista Películas.
Lanzar proceso de actualización de datos y geoposición.	Ok	Se recalcula la posición de usuario y se vuelven a obtener los datos de los proveedores de información al presionar el botón Actualizar.

Tabla 5.1 Pruebas generales sobre la interfaz Kinetea

* La actividad recoge toda la información acerca de los cines que proporciona la página web de google dedicada a ello. El problema está en que en esta página no aparecen todos los cines que se encuentran cerca de la posición de usuario aunque sí la mayoría. Los que no aparecen tienen dos tipos de perfil: salas pequeñas en localidades secundarias o cines que pertenecen a cadenas de multisalas, como por ejemplo Cinesa o Kinepolis, que utilizan otros medios de compra distintos a entradas.com. En los dos casos parece que tiene un marcado carácter comercial y que ha de responder a la existencia o falta de acuerdos entre las diferentes empresas. Aun así, en ocasiones las carteleras de los cines Cinesa aparecen en la página, pero en ningún caso tienen habilitada la función de comprar entradas.

- Mapa:

Prueba	Resultado	Observaciones
Marcadores creados correctamente.	~Ok*	Los marcadores se posicionan en el mapa en su lugar correcto y contienen cada uno la información del evento al que representan. Los eventos de tipo cine se representan con marcadores de color azul y la posición del usuario se señala con un marcador rojo.
Mostrar menú de aplicación.	Ok	Se despliega correctamente el menú de opciones al presionar el botón de Opciones de mapa y trayecto o al presionar el botón de menú del dispositivo.
Cambiar tipo de mapa	Ok	Se despliega correctamente el submenú para cambiar el tipo de mapa y se realizan los cambios.
Cambiar medio de transporte	Ok	Se despliega correctamente el submenú para cambiar el tipo de transporte y se realizan los cambios.
Detalles de transporte sin seleccionar un evento.	Ok	Se muestra un mensaje al usuario indicándole que no se ha seleccionado ningún evento sobre el que mostrar los detalles de la ruta.
Detalles de transporte con evento seleccionado y rutas no calculadas.	Ok	Se muestra una ventana de diálogo al usuario informándole que no existen rutas calculadas para el medio de transporte seleccionado y se le ofrece

		la opción de calcularlo. Una vez calculado muestra la lista de detalles.
Detalles de transporte con evento seleccionado y rutas calculadas.	Ok	Se muestra la actividad de lista de detalles de ruta con la descripción del trayecto paso por paso.
Ventana de información de marcador creada correctamente.	Ok	Marcador asociado al cine correspondiente y que muestra la información de éste en su ventana.
Evento sobre ventana de marcador recogido correctamente.	Ok	Al pulsar sobre la ventana del marcador se muestra una ventana de diálogo con la opción de calcular la ruta hasta la posición del evento o de mostrar la lista de películas.
Mostrar cartelera de cine.	Ok	Se lanza correctamente la actividad que muestra la cartelera del cine seleccionado.
Opción de cómo llegar sin rutas calculadas para el medio de transporte especificado.	Ok	Se calcula la ruta para el medio transporte seleccionado y se pinta la ruta en el mapa.
Opción de cómo llegar con rutas calculadas para el medio de transporte especificado.	Ok	Pintar en el mapa todas las rutas calculadas para la posición de este evento.

Tabla 5.2 Pruebas generales sobre la interfaz Mapa

* En ocasiones las direcciones de los cines que proporciona la página de Google no tiene el formato óptimo para el servicio web de geolocalización y las coordenadas no pueden ser resueltas correctamente. Esto sucede, sobre todo, en cines situados en centros comerciales de ciudades más pequeñas en cuya dirección se indica un determinado kilómetro de una carretera o se añade a la dirección información sobre el polígono o centro comercial. Se intenta tratar el formato para eliminar este tipo de referencias, aunque no responden siempre al mismo patrón. También se intenta hacer la búsqueda por el nombre del cine, que en muchas ocasiones está almacenado en la información de Google y es capaz de ubicarlo. Aun así hay casos en los que no es posible resolver la dirección del cine y no se incluye en el mapa ni en el listado.

- ListaCine:

Prueba	Resultado	Observaciones
Muestra cartelera del cine correctamente y lista desplegable funciona.	Ok	Crea una lista expandible con todas las películas que se están proyectando en el cine y al pulsar sobre cualquiera de ellas aparece una lista con los horarios de las sesiones.
Películas correctamente ordenadas.	Ok	Las películas se listan en orden: primero por género según las preferencias del usuario, después por importancia según las ha clasificado Google en su página y tercero agrupando los diferentes formatos para un mismo título.
Link de horario funciona correctamente.	Ok	Si existe un enlace para comprar la entrada, se lanza correctamente el explorador y se redirige a la página que permite comprar la entrada para esa sesión en concreto.
Si no hay link disponible se informa al usuario.	Ok	Si el link para la sesión no está disponible en la página de Google se informa al usuario con un mensaje.
Botón Cómo llegar calcula y muestra ruta.	Ok	Se calcula trayecto con el tipo de transporte seleccionado en ese momento y se comprueba si la ruta está ya calculada, si no lo está se calcula, y en ambos casos se lanza la actividad Mapa y se pinta el trayecto.
Botón más eventos vuelve al mapa.	Ok	Si se pulsa el botón de + Eventos muestra la actividad del mapa con los marcadores en posición.
Aparece botón de Menú Principal cuando se lanza la actividad desde Mapa o cuando el usuario está muy cerca de un único	Ok	Si se lanza esta actividad desde el mapa, aparece el botón de menú principal y al pulsarlo la aplicación

evento y éste es de tipo cine, y funciona correctamente.		vuelve a la actividad principal.

Tabla 5.3 Pruebas generales sobre la interfaz ListaCine

- ListaDetallesRuta:

Prueba	Resultado	Observaciones
Lista de detalles correcta para ruta calculada.	Ok	Se muestra una lista con los detalles de la ruta que se haya calculado y aparece marcada la pestaña para el medio de transporte correspondiente.
Cambio de pestaña de medio de transporte.	Ok	Cambia de modo de transporte y comprueba si ya se ha calculado la ruta para el medio de transporte seleccionado. Si no, se calcula. Una vez calculado se muestra en la lista y aparece seleccionada la pestaña correspondiente.
Al presionar sobre el botón atrás para volver a mapa, está pintado el camino.	Ok	Cuando se pulsa sobre el botón Atrás se vuelve a la actividad del mapa y aparece pintados todos los trayectos calculados entre el usuario y el evento activo.

Tabla 5.4 Pruebas generales sobre la interfaz ListaDetallesRuta

- Lista películas:

Prueba	Resultado	Observaciones
Películas ordenadas correctamente.	Ok	Las películas se listan en orden: primero por género según las preferencias del usuario, después por importancia según las ha clasificado Google en su página y tercero agrupando los diferentes formatos para un mismo título.
Doble lista expandible funcional.	Ok	Se muestra una lista doble expandible que tiene como lista principal las películas. Al pulsar sobre cada película se despliega una lista con los cines en

		los que se proyecta dicha película. Al pulsar sobre un cine se despliega una lista con los horarios de las sesiones disponibles de esa película en ese cine.
Funcionamiento del enlace para comprar la entrada.	Ok	Si existe un enlace para comprar la entrada, se lanza correctamente el explorador y se redirige a la página que permite comprar la entrada para esa sesión en concreto. Si no existe el enlace, se informa al usuario mediante un mensaje.
Vista de mapa con los cines en los que se representa la película	Ok	Al presionar sobre el botón de ver cines situado en cada elemento película de la lista, se lanza una actividad con un mapa en el que se muestran marcadores con la posición de los cines en los que se representa la película seleccionada.
Vuelta a menú principal al pulsar botón.	Ok	Cuando se pulsa sobre el botón Menú principal se vuelve a la actividad principal.

Tabla 5.5 Pruebas generales sobre la interfaz ListaPeliculas

- PelículasMapa:

Prueba	Resultado	Observaciones
Marcadores creados correctamente.	Ok	Se posicionan únicamente los marcadores correspondientes a los cines en los que se representa la película.
Mostrar menú de aplicación.	Ok	Se despliega correctamente el menú de opciones al presionar el botón de Opciones de mapa y trayecto o al presionar el botón de menú del dispositivo.
Cambiar tipo de mapa	Ok	Se despliega correctamente el submenú para cambiar el tipo de mapa y se realizan los cambios.

Cambiar medio de transporte	Ok	Se despliega correctamente el submenú para cambiar el tipo de transporte y se realizan los cambios.
Detalles de transporte sin seleccionar un evento.	Ok	Se muestra un mensaje al usuario indicándole que no se ha seleccionado ningún evento sobre el que mostrar los detalles de la ruta.
Detalles de transporte con evento seleccionado y rutas no calculadas.	Ok	Se muestra una ventana de diálogo al usuario informándole que no existen rutas calculadas para el medio de transporte seleccionado y se le ofrece la opción de calcularlo. Una vez calculado muestra la lista de detalles.
Detalles de transporte con evento seleccionado y rutas calculadas.	Ok	Se muestra la actividad de lista de detalles de ruta con la descripción del trayecto paso por paso.
Ventana de información de marcador creada correctamente.	Ok	Marcador asociado al cine correspondiente y que muestra la información de éste en su ventana.
Evento sobre ventana de marcador recogido correctamente.	Ok	Al pulsar sobre la ventana del marcador se muestra una ventana de diálogo con la opción de calcular la ruta hasta la posición del evento.
Opción de cómo llegar sin rutas calculadas para el medio de transporte especificado.	Ok	Se calcula la ruta para el medio transporte seleccionado y se pinta la ruta en el mapa.
Opción de cómo llegar con rutas calculadas para el medio de transporte especificado.	Ok	Pintar en el mapa todas las rutas calculadas para la posición de este evento.
Botón de Menú Principal	Ok	Al pulsar el botón de Menú Principal se vuelve a la actividad principal.
Botón de Volver Horarios	Ok	Al pulsar el botón de Volver a Horarios se vuelve a la actividad de lista de películas.

Tabla 5.6 Pruebas generales sobre la interfaz PeliculasMapa

5.2 Pruebas de validación

Para este tipo de pruebas se han de modificar ciertos parámetros del programa para poder simular la ejecución de la aplicación en un caso determinado.

5.2.1 Caso 1: Usuario situado en el país vasco.

Si el usuario está situado en algún lugar del País Vasco, se puede utilizar el servicio web de kulturklik para obtener información sobre las obras de teatro que hay a su alrededor. Para ello simulamos el sistema de localización para ubicar al usuario en un punto del País Vasco y ver la ejecución de la aplicación en este caso.

De nuevo se vuelven a agrupar las pruebas por la actividad en las que se producen:

- Kinetea:

Prueba	Resultado	Observaciones
Cambiar opciones de búsqueda.	Ok	Se accede al submenú de opciones de búsqueda y se permite elegir entre obtener información de eventos de tipo cine, teatro o ambos.
Obtención de eventos de tipo cine únicamente.	Ok	Se obtiene la información únicamente de los cines.
Obtención de eventos de tipo teatro únicamente.	Ok	Se obtiene la información únicamente de los teatros.
Obtención de eventos de tipo cine y teatro.	Ok	Se obtiene información de los dos tipos de eventos.
Opción de listar películas desactivada cuando se han calculado eventos sólo de teatros.	Ok	Si se ha obtenido información únicamente para eventos de tipo teatro, no se muestra la opción de mostrar la lista de películas ya que no existe información al respecto.

Tabla 5.7 Pruebas usuario en País Vasco sobre la interfaz Kinetea

- Mapa:

Prueba	Resultado	Observaciones
Coloca los marcadores correctamente y asigna colores dependiendo de lo que representen.	Ok	Los marcadores se posicionan en el mapa en su lugar correcto y contienen cada uno la información del evento al que representan. Los eventos de tipo cine se representan con marcadores de color azul, los de tipo teatro con marcadores de color verde y la posición del usuario se señala con un marcador de color rojo.
Ventana de información de marcador creada correctamente.	Ok	Marcador asociado al evento correspondiente y que muestra la información de éste en su ventana.
Evento sobre ventana de marcador recogido correctamente.	Ok	Al pulsar sobre la ventana del marcador se muestra una ventana de diálogo con la opción de calcular la ruta hasta la posición del evento y, dependiendo de si es un cine o un teatro, se muestra la opción de ver la cartelera para el primer caso o la página del evento en el segundo.
Mostrar cartelera de cine.	Ok	Se lanza correctamente la actividad que muestra la cartelera del cine seleccionado.
Mostrar página de evento teatro.	Ok	Se abre el explorador por defecto en la página referente a la obra de teatro. En esta página se expone información detallada acerca de la obra y permite acceder a la comprar de entradas.

Tabla 5.8 Pruebas usuario en País Vasco sobre la interfaz Mapa

5.2.2 Usuario cerca de un evento

Si un usuario se encuentra muy cerca de la posición de un evento, la aplicación da prioridad a este evento sobre el resto y muestra su información directamente. Para esto de nuevo se ha de simular la posición del usuario.

- Kinetea:

Prueba	Resultado	Observaciones
Usuario muy cerca de evento cine.	Ok	Después de obtener la información de posición y eventos, se muestra directamente la lista con la cartelera del cine.
Usuario muy cerca de evento teatro.	Ok	Después de obtener la información de posición y eventos, se muestra directamente la página web en la que se encuentra la información de la obra.

Tabla 5.9 Pruebas eventos cercanos sobre la interfaz Kinetea

5.2.3 Varios eventos cerca de la posición de usuario.

Cuando la posición de varios eventos están muy próximos entre ellos y a la posición de usuario, la aplicación no puede decidir con certeza a cuál de los dos tiene pensado asistir el usuario, por lo que se muestran en una lista reducida desde la cual se puede acceder a la información de cada uno. Esto puede suceder, por ejemplo, en la plaza de Callao de Madrid, en la que hay varios cines separados por pocos metros.

Como se ha contemplado que en esta lista de cines cercanos también puedan aparecer teatros, la prueba se va a realizar modificando el criterio de lo que se considera una distancia cercana y ubicando al usuario cerca, según este nuevo criterio provisional, de un teatro y un cine.

Agrupamos las pruebas por actividad:

- Kinetea:

Prueba	Resultado	Observaciones
Usuario muy cerca de varios eventos	Ok	Después de obtener la información de posición y eventos, se detectan varios eventos muy cercanos y se muestra la lista de eventos cercanos.

Tabla 5.10 Pruebas varios eventos cercanos sobre la interfaz Kinetea

- ListaEventosCercanos

Prueba	Resultado	Observaciones
Aparecen correctamente todos los eventos cercanos.	Ok	Se crea la lista de eventos cercanos únicamente con los eventos que han cumplido con la condición de estar a una distancia muy cercana al usuario.
Mostrar cartelera de cine al pulsar sobre evento de tipo cine.	Ok	Si el evento pulsado es de tipo cine, se muestra su cartelera en una lista expandible.
Mostrar página de información de la obra al pulsar sobre evento de tipo teatro.	Ok	Si el evento pulsado es de tipo teatro, Se abre el explorador y se muestra la página de información de la obra.
Botón Más cines	Ok	Se muestra el mapa con todos los marcadores posicionados correspondientes a todos los eventos de los que se ha obtenido información
Botón salir	Ok	Devuelve al usuario a la pantalla del menú principal.

Tabla 5.11 Pruebas varios eventos cercanos sobre la interfaz ListaEventosCercanos

- CineListaExpandible:

Botón Atrás funciona correctamente.	Ok	Si se lanza esta actividad desde la lista de eventos cercanos aparece el botón Atrás en lugar del botón de menú principal. Al pulsarlo se vuelve a la actividad de cines cercanos.

Tabla 5.12 Pruebas varios eventos cercanos sobre la interfaz CineListaExpandible

Capítulo 6. Conclusiones y líneas de trabajo futuras

6.1 Conclusiones

Se ha desarrollado una aplicación móvil para el sistema operativo Android basada en la localización del usuario y adaptativa según contexto, que permite la compra de entradas para eventos.

La idea original era conseguir una aplicación que proporcionase varias opciones de ocio al usuario para ese mismo instante, es decir, que un usuario que se encontrara paseando o tomando algo en un bar, pudiese decidir cuál sería la siguiente actividad a realizar a través de su dispositivo móvil. Esta idea tuvo que ser descartada desde el principio, puesto que no era posible abarcar todas las posibilidades de ocio con los servicios web disponibles al público. Por tanto, se enfocó la búsqueda en eventos en los que se pudiese aportar un valor añadido además de localizarlos. De esta manera se determinó que la funcionalidad principal de la aplicación fuese la de facilitar la compra de entradas para eventos que estén en los alrededores de la posición del usuario.

Bajo estas características se hizo una selección de posibles eventos para incluir en la aplicación, tales como sesiones de cine, obras de teatro, conciertos, eventos deportivos, etc. Se buscaba el requisito de poder obtener información en tiempo real de la red, ya fuera mediante peticiones a servicios web o a través de información extraída de una página de internet.

La búsqueda de fuentes de información no fue sencilla. La mayoría de las páginas que agrupan diferentes tipos de eventos no permiten acceder a su información a través de servicios web. Además las búsquedas que permiten hacer desde sus portales web suelen utilizar funciones que acceden a su base de datos, y no están disponibles a partir de una determinada *url*. Esto hace que la extracción de información a partir del código *html* de la página sea una tarea complicada y costosa computacionalmente.

En vista a la dificultad que suponía encontrar información acerca de eventos, se decidió centrar el foco en los dos tipos más habituales, las obras de teatro y las sesiones de cine.

En lo que concierne a las obras de teatro, se encontró el dominio web de Kulturklik que ofrece información sobre diferentes eventos culturales y permite redirigir al usuario a

la página correspondiente para comprar la entrada. Este servicio depende de la consejería del País Vasco y está únicamente destinado para eventos que se producen en esta comunidad. Además, dispone de un servicio web sobre el que se pueden realizar peticiones en función de una posición dada y una región de búsqueda.

Para los cines no fue posible encontrar un servicio web que proporcionase la información requerida, pero se encontró una página dedicada de Google que ofrece resultados de las carteleras de los cines por ciudad, localidad o posición. Las diferentes opciones que se pueden configurar en la página, se introducen como parámetros en la *url*, por lo que es sencillo construir ésta para que la página muestre los datos que se desean. El formato *html* de la página es sencillo y constante, por lo que es posible extraer toda la información que se necesita.

Ante la falta de información sobre eventos, se barajó la posibilidad de incluir en el proyecto un servidor web accesible desde la aplicación móvil, y a través de éste simular la obtención de los datos. Esta opción se desestimó ya que la funcionalidad y potencial de la aplicación se podía desarrollar con las dos fuentes disponibles.

Para suplir la escasez de fuentes de información y enriquecer la aplicación desde el punto de vista académico, se ha intentado dotar a ésta de la mayor funcionalidad posible. De este modo se ha hecho uso de varias librerías de Android para conseguir diferentes objetivos.

Para implementar la funcionalidad principal se ha hecho uso de los sistemas de geolocalización que permiten determinar la posición del usuario y obtener la información en función de ella. Después se ha utilizado la librería de JSON para comunicar a la aplicación con el servicio web de los teatros y parsear su respuesta. Posteriormente, para poder extraer la información del código *html* de la página web de cines se ha utilizado la librería de JSOUP. Una vez que se han adquirido los datos, se utilizan las librerías de mapas de Google para representarlos mediante marcadores en un mapa interactivo junto a la posición de usuario. Estos marcadores se asocian a los eventos que representan, y permiten el acceso a la página de información, en el caso de una obra de teatro, o a la cartelera, en el caso de un cine.

A esta funcionalidad principal se le han añadido otras. Se ha incluido el uso de otro servicio web, Google Directions, para obtener la información de trayecto entre la posición del usuario y cada uno de los eventos. Se ha incluido la posibilidad de ver la cartelera de los cines por películas, a través de una lista que muestra por cada película los cines en los que se proyecta, y por cada cine los horarios de las sesiones; e incluso visualizar estos cines sobre el mapa. Por último, y para ofrecer una experiencia más personal, se ha incluido unas variables de sesión que almacenan las preferencias del usuario, y que permiten, entre otras cosas, ordenar las películas en cartelera atendiendo a los gustos del usuario.

Con todo esto, aunque no se han podido obtener las fuentes de información que proporcionan toda la información posible sobre todos los eventos, sí que se ha conseguido desarrollar una aplicación funcional y atractiva.

En el siguiente apartado se verán posibles mejoras a realizar sobre la actual versión.

6.2 Líneas Futuras

En este capítulo se va a indicar algunas líneas de desarrollo para ampliar el catálogo de eventos disponibles y la funcionalidad de la aplicación.

6.2.1 Ampliación de catálogo

6.2.1.1 Ampliación catálogo de cines

Como se ha visto en el apartado de pruebas, la página web de www.google.es/movies no dispone de toda la información de todos los cines. Los cines más sencillos o los que corresponden a algunas cadenas como Cinesa o Kinepolis no aparecen en la lista de cines disponibles.

La conclusión ante las faltas de grandes multisalas en el listado ofrecido por Google, es que responde a una motivación comercial que estará basada en acuerdos entre diferentes entidades. Así, todas las entradas que permite comprar la web utilizada en la aplicación se realizan a través del dominio de entradas.com. En cambio, las entradas para los cines del grupo Cinesa se efectúan desde la página de Ticketmaster y los del grupo Kinepolis tienen su propia gestión de las entradas. Una forma de ampliar el catálogo de cines, sería incluir como fuentes de información las páginas web de estas empresas.

Tanto en la página del grupo Cinesa como en el de Kinepolis se puede obtener la información relativa a películas y horarios. Para ello se puede formar una dirección url dentro de su dominio, dependiente del nombre de un cine en particular, para acceder a la información requerida. Una vez identificada la página se puede parsear su contenido para adaptar la información a la estructura de la aplicación.

Para poder incluir estos nuevos datos en la aplicación se debería disponer de la posición y nombre de cada uno de los cines y ejecutar la búsqueda cuando procediese. Para almacenar estos datos se podría hacer mediante un archivo de texto plano de tipo *properties* en los que se indicase el nombre, posición y *url* de consulta. Una vez ubicado el usuario debería comprobar cuáles de los cines se encuentran en su radio de acción y generar una consulta por cada uno. Para ahorrar tiempo de carga inicial, la extracción de la información de la página de cada cine se podría hacer en el momento en el que el usuario seleccionase el marcador correspondiente.

6.2.1.2 Ampliación catálogo teatros

Para poder ampliar el catálogo de teatros a partir de opciones en la red es más complicado.

Una forma de buscar teatros cerca de la posición de usuario es utilizar el servicio web de Google Places. Este sistema, como vimos en el capítulo de estado del arte, permite encontrar diferentes tipos de establecimientos en las proximidades de una determinada ubicación. El problema principal es que no existe la categoría Teatro creada expresamente en este servicio. En cambio se puede hacer la búsqueda por establecimientos genéricos e indicar en la petición que se devuelvan únicamente aquellos que contengan la palabra teatro en su nombre.

Con este sistema se conseguiría una lista de teatros a situar en el mapa aunque no se dispondría de la información relativa a los eventos que se representarían en ellos. Una vez obtenidos los teatros se podría volver a hacer una petición de información de lugar al servicio web de Google Places para cada uno de ellos. En la respuesta se incluye la dirección web que corresponde al teatro y se podría redirigir al usuario hasta esta página. El problema es que muchas de estas páginas no ofrecen toda la información necesaria sobre el evento que se representa actualmente y muchas no incluyen la información necesaria para poder comprar la entrada para el evento.

Al final se conseguiría un catálogo de teatros más amplio pero con el que no se podría ofrecer al usuario una forma sencilla de ver la información del evento y comprar la entrada en muchos de los casos.

Otra forma de ampliar el catálogo de teatros sería la que utilizan la mayoría de las otras aplicaciones de entradas que se han analizado, acuerdos comerciales con las empresas encargadas de los eventos. Habría de contactar con las empresas promotoras que ejercen en los diferentes teatros y pedir la información relativa a sus eventos. Este tipo de acuerdos son habituales ya que por el lado de la empresa promotora permite la difusión del evento y por parte del equipo de la aplicación consigue ampliar el catálogo de eventos y por lo tanto resultar más atractiva para que los usuarios la utilicen.

6.2.1.3 Ampliación de catálogo de otro tipo de eventos

Al igual que en el caso de los teatros, no es sencillo obtener información sobre otro tipo de eventos en la web (conciertos, eventos deportivos, exposiciones, etc…). Para poder ofrecer información al usuario sobre otro tipo de eventos, habría que recurrir de nuevo a los posibles acuerdos comerciales que se puedan hacer entre las empresas promotoras y el grupo de la aplicación.

6.2.2 Ampliación de funcionalidades

6.2.2.1 Inclusión en las redes sociales

En la actualidad el uso de las redes sociales se está introduciendo en todos los ámbitos y se convierte en una herramienta muy útil para compartir opiniones y experiencias con otras personas. Por ello una posible mejora de la aplicación podría ser la posibilidad de conectarse a Facebook o Twiter para compartir o valorar un evento al que se haya asistido, o buscar opiniones o recomendaciones que puedan haber compartido los contactos del usuario sobre un evento determinado.

6.2.2.2 Incluir una lista de obras de teatro

En el supuesto de que se haya aumentado el catálogo de eventos del tipo obras de teatro, se podría incluir una actividad similar a la de listado de películas. En la actividad del listado de películas se muestra una lista ordenada de todas las opciones disponibles en cartelera ordenada según las preferencias del usuario.

Una posible mejora sería llevar ese mismo concepto a las obras de teatro y ofrecer la posibilidad desde el menú principal de acceder a un listado con todas las obras de teatro disponibles ordenadas según las preferencias del usuario. Para ello sería necesario almacenar en la memoria permanente de la aplicación las obras de teatro que han interesado al usuario y en función de ellas crear un ranking de géneros por el cuál se ordenen las opciones disponibles.

6.2.2.3 Crear una actividad de eventos destacados

En todas las aplicaciones de eventos generalistas ya existentes que se han examinado se incluye una selección de eventos destacados. Una posible mejora de la aplicación sería mostrar este tipo de información, creando una lista que mostrase únicamente los eventos más importantes, utilizando las preferencias del usuario para determinar cuáles serían estos eventos.

Así, por ejemplo, la cartelera de los cines que ofrece Google ordena la lista de películas en función de si es estreno, taquillera, etc. Esta información se une a la que se almacenan en la memoria permanente de la aplicación sobre las preferencias de usuario para ordenar los resultados de las películas.

De esta manera se podría extraer de la cartelera los títulos que el usuario pueda considerar más atractivos y juntarlos con otro tipo de eventos considerados importantes obtenidos con un mecanismo similar. Así, se confeccionaría una lista con eventos destacados a la medida del usuario basado en sus preferencias y no en razones comerciales como ocurre en las aplicaciones de venta de entradas que se han examinado.

Bibliografía

[1] "Spain Digital Future in Focus", comScore, 15 de Abril de 2013, Disponible en: http://www.comscore.com/esl/Insights/Presentations_and_Whitepapers/2013/2013_Spain_Digital_Future_in_Focus [Consultado Noviembre 2013]

[2] "Android ya está en 9 de cada 10 nuevos smartphones", Kantar Worldpanel ComTech, 17 de Abril de 2013. Disponible en: http://www.kantarworldpanel.com/es/Noticias/Android-ya-est-en-9-de-cada-10-nuevos-smartphones [Consultado Noviembre 2013]

[3] "IDC Press Release", Worldwide Quarterly Tablet Tracker, 11 de Septiembre de 2013. Disponible en: http://www.idc.com/getdoc.jsp?containerId=prUS24314413 [Consultado Noviembre 2013]

[4] API de KulturKlik. Disponible en: http://www.kulturklik.euskadi.net/lang/es/laguntza-ayuda/api/ [Consultado Noviembre 2013]

[5] Página de cartelera de Google. Disponible www.google.es/movies [Consultado Noviembre 2013]

[6] Página Wikipedia sobre Android, http://es.wikipedia.org/wiki/Android [Consultado Noviembre 2013]

[7] Ingrid Lunden. "Android, Led By Samsung, Continues To Storm The Smartphone Market, Pushing A Global 70% Market Share". Julio 2013. Disponible en: http://techcrunch.com/2013/07/01/android-led-by-samsung-continues-to-storm-the-smartphone-market-pushing-a-global-70-market-share/?ncid=tcdaily [Consultado Noviembre 2013]

[8] Luís Recuenco Pérez: "Desarrollo de una aplicación para un terminal móvil con soporte para geolocalización". Proyecto fin de Carrera, Universidad Carlos III de Madrid, Octubre 2008.

[9] Location Strategies, Andoid Developers. Disponible en: http://developer.android.com/guide/topics/location/strategies.html [Consultado Noviembre 2013]

[10] GPS, http://www.portaleureka.com/accesible/tecnologia/41-gps-donde-estoy [Consultado Noviembre 2013]

[11] GPS, http://es.wikipedia.org/wiki/Sistema_de_posicionamiento_global [Consultado Noviembre 2013]

[12] WPS, http://en.wikipedia.org/wiki/Wi-Fi_positioning_system [Consultado Noviembre 2013]

[13] Imagen WPS, http://mobizen.pe.kr/724 [Consultado Noviembre 2013]

[14] How Google--and everyone else--gets Wi-Fi location data. Disponible en: http://www.zdnet.com/blog/networking/how-google-and-everyone-else-gets-wi-fi-location-data/1664 [Consultado Noviembre 2013]

[15] What is Cell ID?, AT&T Developer Program, Disponible en: http://developer.att.com/developer/tier2page.jsp?passedItemId=3100144 [Consultado Noviembre 2013]

[16] Imagen Cell-ID, http://edu.zdor.cn/edu/0811/201403_3.htm [Consultado Noviembre 2013]

[17] "Mapas en Android (Google Maps Android API v2) – I", Salvador Gómez Oliver. Disponible en: http://www.sgoliver.net/blog/?p=3244 [Consultado Noviembre 2013]

[18] "Mapas en Android (Google Maps Android API v2) – II", Salvador Gómez Oliver. Disponible en: http://www.sgoliver.net/blog/?p=3271 [Consultado Noviembre 2013]

[19] "Mapas en Android (Google Maps Android API v2) – III", Salvador Gómez Oliver. Disponible en: http://www.sgoliver.net/blog/?p=3286 [Consultado Noviembre 2013]

[20] API Google Places, Disponible en: https://developers.google.com/maps/documentation/places/?hl=es [Consultado Noviembre 2013]

[21] API de Google Directions, Disponible en: https://developers.google.com/maps/documentation/directions/?hl=es [Consultado Noviembre 2013]

[22] iGoogle secret API's. Disponible en: http://motyar.blogspot.com.es/2011/11/i-googles-secret-apis.html [Consultado Noviembre 2013]

[23] Pregunta "Is there a movi showtime API?", Stack Overflow, Disponible en: http://stackoverflow.com/questions/439857/is-there-a-movie-showtime-api [Consultado Noviembre 2013]

[24] Página de JSOUP, Disponible en: http://www.jsoup.org [Consultado Noviembre 2013]

[25] Página web de entradas.com. Disponible en: www.entradas.com [Consultado Noviembre 2013]

[26] Página web de Kulturklik, Disponible en: www.kulturklic.euskadi.net [Consultado Noviembre 2013]

[27] Aplicación web atrapalo.com, Disponible en: http://m.atrapalo.com [Consultado Noviembre 2013]

[28] API Android Developers, Activity, Disponible en: http://developer.android.com/reference/android/app/Activity.html [Consultado Noviembre 2013]

[29] API Android Developers, LocationListener, Disponible en: http://developer.android.com/reference/android/app/Activity.html [Consultado Noviembre 2013]

[30] API Android Developers, LocationManager, Disponible en: http://developer.android.com/reference/android/location/LocationManager.html [Consultado Noviembre 2013]

[31] API Android Developers, Geocoder, Disponible en: http://developer.android.com/reference/android/location/Geocoder.html [Consultado Noviembre 2013]

[32] API Android Developers, SharedPreferences, Disponible en: http://developer.android.com/reference/android/content/SharedPreferences.html [Consultado Noviembre 2013]

[33] API Android Developers, AsyncTask, Disponible en: http://developer.android.com/reference/android/os/AsyncTask.html [Consultado Noviembre 2013]

[34] API Android Developers, JSONObject, Disponible en: http://developer.android.com/reference/org/json/JSONObject.html [Consultado Noviembre 2013]

[35] API Android Developers, JSONArray, Disponible en: http://developer.android.com/reference/org/json/JSONArray.html [Consultado Noviembre 2013]

[36] API Android Developers, HashMap, Disponible en: http://developer.android.com/reference/java/util/HashMap.html[Consultado Noviembre 2013]

[37] API Android Developers, LinkedHashMap, Disponible en: http://developer.android.com/reference/java/util/LinkedHashMap.html [Consultado Noviembre 2013]

[38] API Android Developers, Polyline, Disponible en: http://developer.android.com/reference/com/google/android/gms/maps/model/Polyline.htm [Consultado Noviembre 2013]

[39] API Android Developers, FragmentActivity, Disponible en: http://developer.android.com/reference/android/support/v4/app/FragmentActivity.html [Consultado Noviembre 2013]

[40] API Android Developers, MapView, Disponible en: http://developer.android.com/reference/com/google/android/gms/maps/MapView.html [Consultado Noviembre 2013]

[41] API Android Developers, Marker, Disponible en: http://developer.android.com/reference/com/google/android/gms/maps/model/Marker.html [Consultado Noviembre 2013]

[42] API Android Developers, Bundle, Disponible en: http://developer.android.com/reference/android/os/Bundle.html[Consultado Noviembre 2013]

[43] API Android Developers, LatLngBounds.Builder, Disponible en: http://developer.android.com/reference/com/google/android/gms/maps/model/LatLngBounds.Builder.html [Consultado Noviembre 2013]

[44] API Android Developers, LatLngBounds, Disponible en: http://developer.android.com/reference/com/google/android/gms/maps/model/LatLngBounds.html [Consultado Noviembre 2013]

[45] API Android Developers, CameraUpdateFactory, Disponible en: http://developer.android.com/reference/com/google/android/gms/maps/CameraUpdateFactory.html [Consultado Noviembre 2013]

[46] API Android Developers, PolilyneOptions, Disponible en: http://developer.android.com/reference/com/google/android/gms/maps/model/PolylineOptions.html [Consultado Noviembre 2013]

[47] API Android Developers, InfoWindowAdapter, Disponible en: http://developer.android.com/reference/com/google/android/gms/maps/GoogleMap.InfoWindowAdapter.html [Consultado Noviembre 2013]

[48] API Android Developers, ExpandableListActivity, Disponible en: http://developer.android.com/reference/android/app/ExpandableListActivity.html [Consultado Noviembre 2013]

[49] API Android Developers, SimpleExpandableListAdapter, Disponible en: http://developer.android.com/reference/android/widget/SimpleExpandableListAdapter.html [Consultado Noviembre 2013]

[50] API Android Developers, ListActivity, Disponible en: http://developer.android.com/reference/android/app/ListActivity.html [Consultado Noviembre 2013]

[51] API Android Developers, ArrayAdapter, Disponible en: http://developer.android.com/reference/android/widget/ArrayAdapter.html [Consultado Noviembre 2013]

[52] API Android Developers, MeasureSpec, Disponible en: http://developer.android.com/reference/android/view/View.MeasureSpec.html [Consultado Noviembre 2013]

[53] "Localización geográfica en Android (II)", Salvador Gómez Oliver. Disponible en http://www.sgoliver.net/blog/?p=1932 [Consultado Noviembre 2013]

[54] "Integrating Google Maps in Android App", Chupa Team. Disponible en: http://www.chupamobile.com/tutorial/details/53 [Consultado Noviembre 2013]

[55] "Mapas en Android (III): Overlays (Capas)", Salvador Gómez Oliver. Disponible en: http://www.sgoliver.net/blog/?p=2004 [Consultado Noviembre 2013]

[56] "Servicios web (1): SOAP no, gracias" , Enrique Amodeo, 19 de Julio de 2010,. Disponible en: http://eamodeorubio.wordpress.com/2010/07/19/servicios-web-1-soap-no-gracias/ [Consultado Noviembre 2013]

[57] SOAP, Wikipedia. Disponible en: http://es.wikipedia.org/wiki/Simple_Object_Access_Protocol [Consultado Noviembre 2013]

[58] "Introducción a SOAP", Warner Onstine, ArcMind. Disponible en: http://www.ibm.com/developerworks/ssa/webservices/tutorials/ws-soa1/section4.html [Consultado Noviembre 2013]

[59] "Servicios web (2): ¿Qué es REST?", Enrique Amodeo, 26 de Julio de 2010. Disponible en: http://eamodeorubio.wordpress.com/2010/07/26/servicios-web-2-%C2%BFque-es-rest/ [Consultado Noviembre 2013]

[60] "Extensible Markup Language (XML)", W3C. Disponible en: http://www.w3.org/XML/ [Consultado Noviembre 2013]

[61] Standard ECMA-262 3rd Edition, Diciembre 1999. Disponibe en: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf [Consultado Noviembre 2013]

[62] Introducing JSON. . Disponible en: http://www.json.org/ [Consultado Noviembre 2013]

[63] "Get the Android SDK", Andorid Developers, Disponible en: http://developer.android.com/sdk/index.html [Consultado Noviembre 2013]

[64] "Setting Up the ADT Bundle", Andorid Developers, Disponible en: http://developer.android.com/sdk/installing/bundle.html [Consultado Noviembre 2013]

[65] "Setting Up an Existing IDE", Andorid Developers, Disponible en: http://developer.android.com/sdk/installing/index.html [Consultado Noviembre 2013]

[66] “Installing the Eclipse Plugin”, Andorid Developers, Disponible en: http://developer.android.com/sdk/installing/installing-adt.html [Consultado Noviembre 2013]

[67] “Adding Platforms and Packages”, Andorid Developers, Disponible en: http://developer.android.com/sdk/installing/adding-packages.html [Consultado Noviembre 2013]

[68] “SDK Manager”, Andorid Developers, Disponible en: http://developer.android.com/tools/help/sdk-manager.html [Consultado Noviembre 2013]

[69] “Using the Emulator”, Andorid Developers, Disponible en: http://developer.android.com/tools/devices/emulator.html [Consultado Noviembre 2013]

[70] “Running Google Maps v2 on Android Emulator”, Kirit Vaghela. Disponible en: http://stackoverflow.com/questions/14040185/running-google-maps-v2-on-android-emulator [Consultado Noviembre 2013]

Anexo A. Presupuesto

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Javier Vaíllo

2.- Departamento:
Ingeniería Telemática

3.- Descripción del Proyecto:
- Titulo
- Duración (meses) 12
Tasa de costes Indirectos: 21%

4.- Presupuesto total del Proyecto (valores en Euros):
25.607,47 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
					0,00	
Florina Almenares		Ingeniero Senior	1	4.289,54	4.289,54	
Javier Vaíllo Martín		Ingeniero	6	2.694,39	16.166,34	
					0,00	
					0,00	
		Hombres mes	**7**	**Total**	20.455,88	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador PC Intel Core 2 Quad	540,00	50	12	60	54,00
Móbil HTC Legend	399,9	100	8	60	53,32
		100		60	0,00
		100		60	0,00
		100		60	0,00
					0,00
				Total	107,32

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Conexión Internet	Orange	600,00
	Total	600,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	20.456
Amortización	107
Subcontratación de tareas	0
Costes de funcionamiento	600
Costes Indirectos	4.444
Total	25.607

Anexo **B.** Google Maps Android v1

Para representar un mapa de Google Maps en una aplicación Android se dispone de la librería com.google.android.maps. La librería no está incluida por defecto en las librerías de Android, por lo que se ha de añadir utilizando el SDK para descargar el paquete Google APIs by Google, dentro del cual viene incluida [54].

Para poder utilizar este paquete se necesita una clave proporcionada por Google, que estará asociada al certificado con el que se firma digitalmente la aplicación. Para pruebas se puede solicitar una clave provisional que habría que cambiar antes de subir una aplicación al *market*.

Este paquete incluye la clase MapActivity de la cual deberán de heredar las actividades que usen un mapa en vez de hacerlo de la clase Activity como se hace comúnmente.

En la clase MapActivity se utiliza el controlador MapView que extiende de la clase ViewGroup de la librería estándar andorid.view. El MapView es el elemento que va a permitir representar en una aplicación un mapa con el diseño de Google Maps. Cuando declaremos este controlador en el *layout* se debe incluir la propiedad *apiKey* y asignarle el valor de la clave que se ha obtenido anteriormente. El MapView implementa la funcionalidad básica para que el usuario pueda hacer zoom y moverse a través del mapa por medio de la pantalla táctil. Si se quiere mostrar explícitamente los controles de zoom estándar sobre el mapa, se puede hacer con el método setBuiltInZoomControls() ejecutado sobre una instancia del MapView.

El mapa básico se puede configurar para mostrarse en tres tipos de vista diferentes. Cada vista tiene un método para activar/desactivar:

- setSatellite(boolean): Habilita la vista satélite que carga fotografías aéreas con las calles y sus nombres superimpresos.
- setTraffic(boolean): Método que muestra el estado del tráfico (no está disponible en toda la geografía española).
- setStreetView(boolean): Método que muestra la vista a pie de calle del lugar.

Para desplazar la vista hacia a una ubicación en el mapa debemos crear primero un objeto de la clase MapController que se encarga de manejar el zoom y el desplazamiento.

Después debemos crear un objeto de la clase Geopoint, perteneciente a la misma biblioteca, que va a tener como atributos las coordenadas geográficas del punto que se quiere ubicar. Estas coordenadas están expresadas en microgrados y se almacenan en formato de entero, por lo que hay que multiplicar las coordenadas expresadas en grados por 1e6.

Con el método animateTo(Geopoint) de la clase MapController hacemos que el mapa se desplace a la ubicación deseada. También se puede elegir el nivel de zoom con el que se visualizará el punto. Este nivel de zoom va a ser un número entero perteneciente al intervalo [1, 21] siendo 1 el de menor detalle y 21 el de mayor.

Una de los usos típicos que se hace del mapa, es la de insertar marcadores para resaltar lugares de interés. Esta funcionalidad se realiza a través de objetos de la clase OverlayItem que permite mostrar un icono para representar una ubicación. Esta clase contiene una serie de atributos en los cuales se va a guardar la información de la ubicación:

- Dispone de tres bits indicadores para monitorizar el estado en el que se encuentra:
    - ITEM_STATE_FOCUSED_MASK: Indica si el control del usuario está sobre el objeto.
    - ITEM_STATE_PRESSED_MASK: Indica si el usuario ha presionado el icono que representa al objeto.
    - ITEM_STATE_SELECTED_MASK: Indica si el usuario ha seleccionado el objeto.
- mMarker: Establece el tipo de icono que se utiliza para representar el objeto en el mapa
- mPoint: objeto Geopoint que indica la geolocalización de la ubicación.
- mSnippet: Texto que se le quiere asignar al objeto para guardar información sobre el punto.
- mTitle: Título con el cuál se quiere identificar al objeto.

El objeto OverlayItem contiene la información del punto, pero necesitamos de otra clase, ItemizedOverlay, que se encargue de manejarlo.

Esta segunda clase agrupa objetos de la clase OverlayItem y se encarga de pintarlos en el mapa con el zoom y el desplazamiento adecuado para que se puedan mostrar todos los elementos a la vez. También es la encargada de gestionar los cambios de actividad en los elementos y enviar esta información de evento a una posible aplicación oyente.

Esta clase está pensada para realizar estas funciones, pero no se suele utilizar directamente. Para poder implementar las acciones y eventos deseados por el programador se crea una clase personalizada que herede de esta y así se puede desarrollar funcionalidades según la necesidad.

Lo primero que hay que implementar en la clase hija es la forma en que se almacenan los objetos OverlayItem. La forma más sencilla es incluir un atributo ArrayList. Hay que crear los métodos que permiten añadir y eliminar elementos de la lista. Cada vez que se haga una modificación en la lista se ha de llamar al método de la clase padre populate()

(este método está protegido y no se puede sobrescribir). Este método se encarga de actualizar los datos y de pintar correctamente los puntos en la pantalla.

Al ejecutarse el método populate() se llama al método createItem(int) para referenciar a cada elemento OverlayItem. Este método se debe sobrescribir para hacerlo compatible con la estructura que se ha elegido para almacenar los objetos OverlayItem. Lo mismo ocurre con el método size() que devuelve el número de elementos que se tiene almacenados.

Finalmente se sobrescriben los métodos que permiten añadirle funcionalidad a los marcadores que aparecen en el mapa. Así uno de los métodos a implementar sería el onTap(int index) que se ejecuta cuando el usuario pulsa sobre uno de los marcadores.

Si se desea incluir otro tipo de información personalizada sobre un mapa, se deben añadir nuevas capas (*overlays*) sobre el controlador MapView. En estas capas es donde se dibuja toda la información adicional [55].

El primer paso para definir una nueva capa de información será crear una clase java que derive de la clase Overlay. En esta nueva clase se sobrescribe el método draw() que es donde se incluye toda la información que se quiere dibujar sobre el mapa. Varias capas pueden subsistir sin problemas sobre un mismo mapa.

El método draw() recibe como parámetro un objeto Canvas, del paquete android.graphics, que tiene varios métodos que permiten dibujar directamente sobre la pantalla: drawLine(), drawCircle(), drawText(), drawBitmap(), etc. Estos métodos no son específicos de los mapas y por lo tanto hay que indicarles las coordenadas en *pixels* relativos a los bordes del control sobre el que se va a dibujar. Para poder crear una equivalencia entre latitudes y longitudes del mapa y los pixeles que ocupan en pantalla se utiliza la clase Projection. Esta clase permite hacer conversiones precisas entre ambos sistemas de referencia.

Para utilizar la clase Projection se crea un objeto GeoPoint a partir de los valores en microgrados de la longitud y latitud de la coordenada que se desee. Se obtiene el objeto Projection mediante el método getProjection() de la clase MapView (parámetro del método draw()). El objeto Projection tiene en cuenta la posición sobre la que está centrada el mapa y el nivel de zoom para realizar la conversión entre latitud-longitud y coordenadas x-y en pixeles. La conversión se realiza a través del método toPixels() que devolverá el resultado sobre un objeto Point de salida. Con objetos de tipo Point ya se pueden usar los métodos de Canvas. Un ejemplo de código sería:

```
public class OverlayMapa extends Overlay {
    private Double latitud = 37.40*1E6;
    private Double longitud = -5.99*1E6;

    @Override
    public void draw(Canvas canvas, MapView mapView, boolean shadow)
    {
        Projection projection = mapView.getProjection();
```

```
            GeoPoint geoPoint =
                new GeoPoint(latitud.intValue(), longitud.intValue());

            if (shadow == false)
            {
                Point centro = new Point();
                projection.toPixels(geoPoint, centro);

                //Definimos el pincel de dibujo
                Paint p = new Paint();
                p.setColor(Color.BLUE);

                //Marca Ejemplo 1: Círculo y Texto
                canvas.drawCircle(centro.x, centro.y, 5, p);
                canvas.drawText("Sevilla", centro.x+10, centro.y+5, p);
            }
        }
}
```

Una vez que se ha definido la capa personalizada se ejecuta el método getOverlays() sobre la instancia de MapView para obtener la lista de capas del mapa. Una vez obtenida la lista se crea una instancia de la capa personalizada y se añade a la lista anterior con el método add(). Finalmente se llama al método postInvalidate() para redibujar el mapa y todas sus capas:

```
mapa = (MapView)findViewById(R.id.mapa);

List<Overlay> capas = mapa.getOverlays();
OverlayMapa om = new OverlayMapa();
capas.add(om);
mapa.postInvalidate();
```

El resultado sería:

Figura B.1 Representación de polígonos y marcadores en mapa

Anexo C. Web Services

C.1 Introducción:

Un servicio web (en inglés, *Web Service* o *Web services*) es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones a través de la red. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos a través de Internet u otro tipo de red.

Para conseguir esta interoperabilidad, los servicios web usan estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

Cada servicio web tiene una interfaz bien definida a través de la cual recibe las peticiones y devuelve las respuestas. Esta interfaz es lo único que verá el cliente y es independiente del lenguaje o mecanismos que se utilizan para formar la respuesta a partir de los datos de entrada que se provee. De esta manera se aporta una gran independencia entre la aplicación que usa el servicio web y el propio servicio, ya que es posible cambiar toda la lógica que se encuentra por debajo de la interfaz manteniendo ésta y hacerlo de una forma totalmente transparente para el usuario. De la misma manera se pueden añadir funcionalidades nuevas al interfaz y seguir utilizando la misma lógica para las funcionalidades que no han cambiado. Esta flexibilidad permite construir grandes aplicaciones a partir de componentes distribuidos más pequeños [56].

Existen diferentes tecnologías para construir servicios web, la más popular es la pila WS-* que es un conjunto de protocolos y estándares para realizar servicios web basados en XML. De éstos los más utilizados son SOAP y WSDL. Posteriormente, y ante la complicación y falta de interoperabilidad que suponen los anteriores protocolos, aparecen otro tipo de tecnologías llamadas REST que se basan en el protocolo HTTP para acceder a los recursos del servicio.

C.2 Tecnologías para servicios web:

C.2.1 SOAP

SOAP (*Simple Object Access Protocol*) es un mecanismo que permite enviar y recibir documentos XML independientemente del protocolo de transmisión o de la estructura del documento XML enviado. Deriva de un protocolo creado por David Winer en 1998,

llamado XML-RPC. SOAP fue creado por Microsoft, IBM y otros y está actualmente bajo el auspicio de la W3C [57].

La especificación SOAP define dos maneras diferentes de emplear SOAP. En primer lugar, SOAP se puede usar para describir un documento XML genérico. A este formato de mensaje SOAP se lo conoce como estilo de mensajería o de documento. En segundo lugar, existe un formato SOAP más específico que exige que el documento XML anidado siga la semántica RPC. Un mensaje SOAP al estilo RPC describe una llamada a procedimiento con su nombre y valores de parámetro o una devolución de procedimiento [58].

La transmisión de un mensaje SOAP involucra tres roles principales:

- El remitente SOAP crea y envía un mensaje SOAP a un receptor SOAP final.
- Se puede posicionar un intermediario SOAP opcional para que intercepte un mensaje SOAP entre un remitente SOAP y un receptor SOAP final. Los intermediarios que interceptan un mensaje SOAP pueden analizarlo a fin de realizar determinadas acciones, como filtrar, registrar, almacenar en caché, etc. antes de enviar el mensaje al destino SOAP final. Al intermediario SOAP se lo puede considerar un remitente-receptor.
- El destino deseado del mensaje SOAP generado por el remitente SOAP (que no es un intermediario) se denomina receptor SOAP final.

Una característica importante de SOAP es que es neutral con respecto a la infraestructura. Esto permite invocar procedimientos remotos a través de distintos protocolos, como por ejemplo SMTP, MQSeries, y como no, HTTP; pero a su vez evita que éste pueda aprovechar cualquier ventaja de la infraestructura web.

C.2.2 WSDL

WSDL (*Web Services Description Language*) es un formato XML que te permite describir, de forma declarativa, el modo de acceso al servicio. El WSDL es el documento o contrato que debe distribuirse a todos los clientes para que éstos tengan toda la información necesaria que les permita este acceso. También es usado por los *frameworks* de publicación de servicios web y por las herramientas de generación de código, tanto a nivel cliente como servidor [56].

En un documento WSDL hay que definir varios artefactos: tipos de datos, mensajes, operaciones, *portTypes, bindings, ports y services*. Los tipos de datos se definen con XML *Schema*, lo que supone una novedad con respecto a SOAP que sólo usa XML.

- Los mensajes definen los documentos XML que van a moverse entre el cliente y el servidor. Para definir un mensaje hay que especificar las partes o secciones de las que se conforma éste y el tipo de datos de cada una de estas secciones.
- Las operaciones definen parámetros de entrada y salida y posibles excepciones. Cada parámetro y excepción debe corresponderse con un mensaje.
- El *portType* es la interfaz del servicio, es decir, el conjunto de operaciones que soporta el servicio.
- Los *bindings* definen el protocolo a utilizar y cómo se van a codificar los mensajes para cada mensaje y operación.
- El *port* define la dirección donde se va a publicar cada *binding*, dirección que depende del protocolo de transporte usado. En caso de usar HTTP, será una URI.
- Finalmente el servicio es el contenedor de todos los *ports* disponibles.

En un WSDL hay mucha información y muy compleja, lo que genera mucho acoplamiento entre el consumidor y el proveedor del servicio.

C.2.3 REST

REST (*Representational State Transfer*) define una familia de arquitecturas con una determinada serie de requisitos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes [59].

REST emergió en los últimos años como el modelo predominante para el diseño de servicios. De hecho, REST logró un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.

Los requisitos de diseño que definen a un servicio web REST son:

- En arquitecturas REST, los servicios no publican un conjunto arbitrario de métodos u operaciones.
- En REST lo que se publica son recursos. Un recurso se puede considerar como una entidad que representa un concepto de negocio que puede ser accedido públicamente.
- Cada recurso posee un identificador único y global, que lo distingue de cualquier otro recurso, aunque ambos tuvieran exactamente los mismos datos.
- Cada recurso posee un estado interno, que no puede ser accedido directamente desde el exterior. Lo que sí es accesible desde el exterior es una o varias representaciones de dicho estado. Por representación se entiende un formato de datos concreto usado para la transferencia de una copia del estado público del recurso entre el cliente y el servidor. La implementación del recurso decide qué

información es visible o no desde el exterior, y que representaciones de dicho estado se soportan.

- En REST todos los recursos comparten una interfaz única y constante. Todos los recursos tienen las mismas operaciones. Las operaciones nos permiten manipular el estado público del recurso. En un sistema REST típico se definen cuatro operaciones:
    - CREATE. En esta operación el cliente manda al servidor una petición para crear un nuevo recurso. Opcionalmente el cliente puede mandar una representación del estado inicial de este recurso. El servidor responde con el identificador global del nuevo recurso.
    - DELETE. En esta operación el cliente elimina un recurso del servidor. El cliente necesita saber el identificador del recurso.
    - READ. Con esta operación el cliente puede leer una representación del estado de un recurso, identificado con su identificador global. El cliente puede especificar qué tipos de representaciones puede interpretar. Lo que sucede realmente es que se copia el estado del recurso en el servidor y se pega en el cliente. Ambas copias del estado no se mantiene sincronizadas. El servidor puede cambiar el estado real del recurso y el cliente, de forma independiente, puede modificar su copia local del estado del recurso.
    - UPDATE. Como el servidor y el cliente tienen una copia diferente del estado, el cliente puede usar esta operación para sobrescribir o grabar su copia del estado en el servidor. De esta manera se puede actualizar el estado del recurso con las modificaciones hechas en el cliente.
- La implementación del servicio es libre de prohibir alguno de estos métodos para un recurso en concreto. También debe definir el modelo de datos que se va a publicar y que representaciones soporta. Para servicios web, los formatos de salida más utilizados son XML y JSON.
- Los distintos recursos se pueden interrelacionar y referenciar entre sí mediante sus identificadores globales.

C.3 Formatos para encapsular la información

En esta aplicación se han utilizado servicios web del tipo REST a los que se le hacía una petición a través de la operación GET del protocolo HTTP para obtener una determinada respuesta. Algunos de estos servicios web permitían escoger el tipo de formato en el que se devolvían los datos en la respuesta a la consulta. Los tipos de formato entre los que se podía escoger son los dos más utilizados globalmente para la encapsulación de información en este tipo de servicios, XML y JSON.

C.3.1 XML

XML (*eXtensible Markup Language*) es un lenguaje de marcas desarrollado por el *World Wide Web Consortium* (W3C) utilizado para almacenar datos en forma legible. Deriva del lenguaje SGML y permite definir la gramática de lenguajes específicos para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones se deben comunicar entre sí o integrar información [60].

XML no sólo se aplica en Internet, es un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo, etc. y permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

XML expresa la información de forma estructurada de manera que sea fácilmente abstraíble y reutilizable. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Esto conforma un árbol de información. A cada una de las partes del documento se le llama elemento, y se las señala mediante etiquetas.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma <nombre>, donde nombre es el nombre del elemento que se está señalando. Las etiquetas definen el principio y el final de un elemento, y lo que vaya entre ellas son los datos ligados a esta etiqueta.

Entre las ventajas de la utilización del XML tenemos que es escalable, ya que permite la adición de nuevas etiquetas sin complicación alguna; se puede utilizar un analizador estándar para extraer la información que contenga y el nombrado de sus etiquetas ofrece información, junto a un contexto, sobre los datos que almacena.

Por otro lado, el uso de XML en los lenguajes de programación o bases de datos, puede resultar en una estructura de árbol básico del XML excesivamente complejo.

C.3.2 JSON

JSON (*JavaScript Object Notation*) es un formato ligero para el intercambio de datos. Tiene una forma de estructurar información sencilla que es fácilmente legible para las personas y que permite que se pueda parsear y generar fácilmente en una máquina. Está basado en una subcategoría del estándar ECMA-262, *JavaScript Programming Language* de diciembre del 1999 [61].

JSON es un formato de texto totalmente independiente del lenguaje con el que se utilice, pero usa ciertas convenciones similares a las que se usan los lenguajes de programación con más penetración. Esto hace que su utilización se haya extendido mucho para el intercambio de información entre programas [62].

Utiliza dos tipos de estructuras:

- Una colección de pares nombre / valor. Define un objeto
- Una lista ordenada de valores. Define un array

Este tipo de estructuras de datos son de uso universal y todos los lenguajes de programación modernos los implementan de una forma u otra.

Formato de objeto y *array* en JSON:

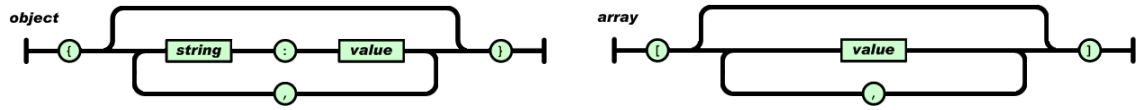

Figura C.1 Estructura del los objetos JSON

JSON se posiciona como una alternativa a XML en los servicios web que tienen una gran cantidad de usuarios por su mayor simplicidad en la estructura y porque el almacenado de los datos ocupa menos espacio.

Anexo D. Instalación entorno

Es este anexo se va a explicar cómo instalar el entorno de desarrollo de Android en un equipo para desarrollar aplicaciones para este sistema operativo.

Existen varios entornos de desarrollo para aplicaciones Android, para este proyecto se ha utilizado Eclipse y será sobre el que trate este anexo.

D.1 Instalación del JDK

Lo primero que se ha de hacer es comprobar que se disponga de una versión del JDK (*Java Development Kit*) de Java actualizada, si no, se puede descargar de la página de Oracle:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Una vez actualizado el JDK de Java existen dos opciones. La primera es descargar el Android SDK (*Software Development Kit*) completo, que es un IDE (*Integrated Development Environment*) de Eclipse que ya tiene incluida el ADT (*Android Developer Tools*). La segunda, es incluir el ADT en un IDE de Eclipse anteriormente instalado en el equipo.

D.2 Instalación del Android SDK

Si se escoge la primera opción, se puede descargar el Android SDK desde la siguiente dirección: [63]

http://developer.android.com/sdk/index.html

Dentro de esta descarga se incluyen todas las librerías y herramientas de desarrollo para crear, probar y depurar aplicaciones para Android. Así se puede encontrar

- Eclipse + ADT *plugin*
- Herramientas de SDK de Android
- Herramienta de plataforma de Android
- La última versión de la plataforma de Android
- La última versión del sistema Android para el emulador.

Una vez descargado el archivo, que tendrá un nombre similar a adt-bundle-<os_platform>.zip, se descomprime en el directorio que se desee. Se creará una carpeta

con el nombre adt-bundle-<os_platform>/eclipse/ en la que se encontrará el archivo ejecutable de Eclipse que permite lanzar el IDE. Es importante no mover los archivos pertenecientes a esta carpeta ya que el sistema no podría encontrarlos y el entorno no funcionaría correctamente [64].

De esta manera ya se tiene el entorno de desarrollo preparado para comenzar a desarrollar aplicaciones.

D.3 Instalación del ADT sobre un IDE de Eclipse existente

Si se escoge la segunda opción y se decide instalar el ADT sobre un IDE de Eclipse existente en el equipo se deben de realizar otra serie de pasos. El primero es descargar el ADT de la dirección:

http://developer.android.com/sdk/index.html#download

Que contiene un archivo ejecutable que lanza un asistente de instalación. El instalador comprueba que esté disponible en el equipo el JDK actualizado y otras herramientas necesarias y descarga e instala lo que no encuentre. Después comienza la instalación el *Android SDK Tool*s en el directorio por defecto o en el que le indique el usuario. Es importante recordar el lugar en el que se guarda el SDK en el equipo, ya que es necesario hacer una referencia a él posteriormente [65].

Una vez se haya completado la instalación aparece una ventana de diálogo ofreciendo abrir el *Android SDK Manager*. En el caso de utilizar Eclipse se debe indicar que no, y en su lugar dirigirse a Eclipse para instalar el plugin correspondiente [66].

Para añadir este complemento se deben de seguir los siguientes pasos:

1. Abrir Eclipse y seleccionar: Help > Install New Software
2. Se ha de rellenar los campos Name y Location en la ventana de diálogo Add Repository que aparece. Se introduce ADT Plugin como nombre y https://dl-ssl.google.com/android/eclipse/ para la dirección.
3. Pulsar Ok. Si se tiene dificultades para descargar el plugin, substituir http por https en la dirección url.
4. En la ventana de Available Software, seleccionar Developer Tools y pulsar Next.
5. En la siguiente pantalla se muestran una serie de herramientas para descargar. Pulsar Next de nuevo.
6. Leer y aceptar los acuerdos de licencia y pulsar Finish. Si aparece alguna ventana de precaución de seguridad indicando que no se puede establecer la autenticidad del software, pulsa Ok

7. Una vez completada la instalación reiniciar Eclipse

Ya se ha instalado el complemento de Android en el entorno de Eclipse, ahora se debe configurar. Para ello, al reiniciar eclipse se debe especificar el lugar en el que se encuentra el directorio del SDK de Android [67]:

1. En la ventana que aparece con el texto Welcome to Android Development, seleccionar Use existing SDKs
2. Navega hasta la carpeta en la que se encuentra el directorio en el que se instaló el Android SDK en el paso anterior.
3. Pulsar Next

El IDE de Eclipse está ahora configurado para desarrollar aplicaciones Android, pero se necesita añadir las últimas versiones de las herramientas de las plataformas de SDK y Android. En el SDK se separan herramientas, plataformas y otros componentes en paquetes que se pueden descargar utilizando el SDK Manager. Para ello se deben seguir los siguientes pasos: [68]

1. Abre el SDK Manager, el archivo SDK Manager.exe se encuentra en la carpeta raíz del directorio del SDK
2. En la figura D.1 se puede observar el aspecto del SDK Manager, que muestra todos los paquetes del SDK que se encuentran disponibles para el Android SDK instalado en el equipo. Como se ha dicho antes, existen diferentes tipos de paquetes y no todos son necesarios en todos los tipos de aplicaciones. Hay una serie de paquetes que se recomiendan instalar, son:
    - La última versión del paquete de Herramientas (carpeta Tools)
    - La última versión del paquete de Android (en la primera carpeta de Android)
    - La librería de compatibilidad de Android (Dentro de la carpeta de Extras seleccionar Android Support Library)

    Una vez escogidos los paquetes que se desea instalar se pulsa sobre el botón de Install para que el SDK Manager instale los paquetes seleccionados en el entorno de desarrollo.

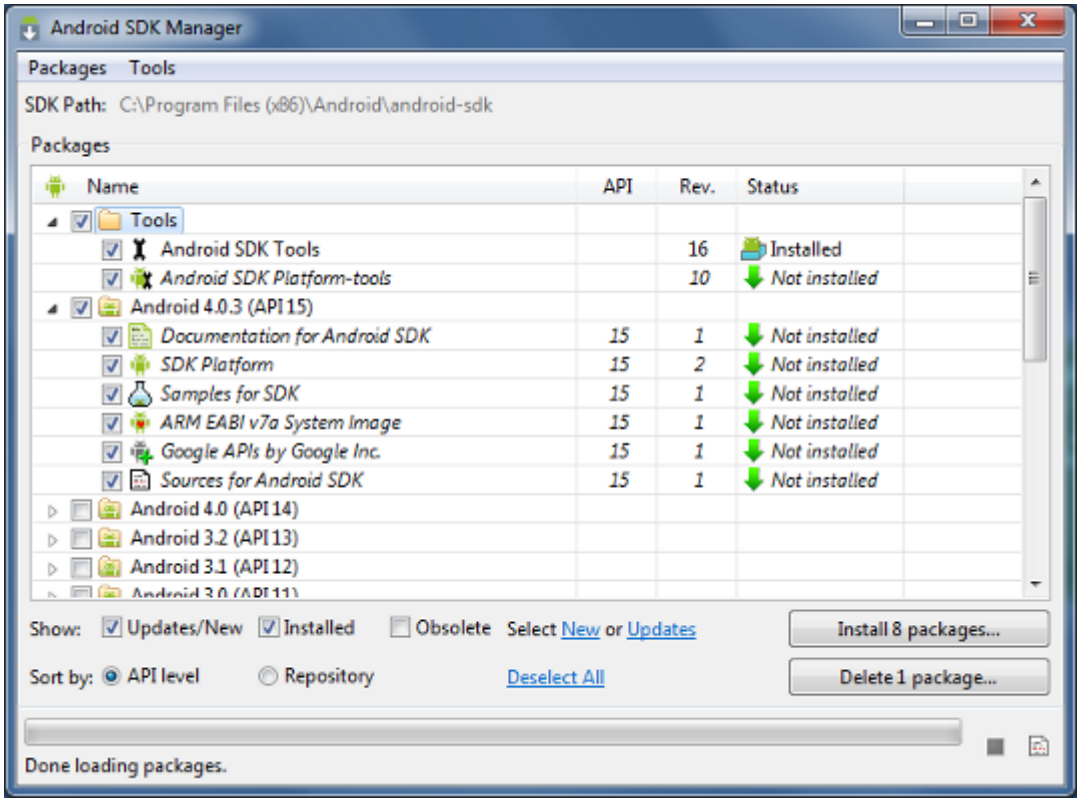

Figura D.1 SDK Manager © [68]

D.4 Instalación del paquete de Google Play para el uso de mapas

Una vez que se ha configurado el entorno de trabajo con los elementos básicos para desarrollar cualquier tipo de aplicación Android, hay que descargar los paquetes específicos que sean necesarios para desarrollar una determinada funcionalidad de la aplicación [17].

En el caso de la aplicación que se ha detallado en este proyecto, era necesario descargar del SDK el paquete de *Google Play Services* que se encuentra dentro de la carpeta de Extras. De nuevo se abriría el *SDK Manager* como se ha detallado en el punto anterior y se puede ver en la Figura C.1, se selecciona el paquete que se ha indicado y se pulsa sobre el botón Install para comenzar la descarga e instalación del paquete en el entorno.

Tras aceptar la licencia correspondiente el paquete quedará instalado en la ruta del sistema: <carpeta-sdk-android>/extras/google/google_play_services/.

Esta dirección hará falta más adelante.

El siguiente paso será obtener una *API Key* para poder utilizar el servicio de mapas de Google en la aplicación. La nueva API de mapas de Android está integrada en la Consola de APIs de Google, por lo que el primer paso será acceder a ella en la dirección:

https://code.google.com/apis/console/

Es necesario disponer de una cuenta de Google. Una vez se ha accedido, se crea un nuevo proyecto desplegando el menú superior izquierdo como se muestra en la figura D.2, y se selecciona la opción Create... .

Al seleccionar la opción aparecerá la ventana mostrada en la figura D.3 solicitando el nombre del proyecto. Se introduce uno que sea descriptivo y se crea el proyecto.

Después, se ha de acceder a la opción Services del menú izquierdo. Desde esta ventana es posible activar o desactivar cada uno de los servicios de Google que se desea utilizar. Para el desarrollo de la aplicación sólo interesa activar el servicio llamado *Google Maps Android API v2*. Para activarlo se pulsa sobre el botón ON/OFF situado justo a su derecha como se ve en la figura D.4.

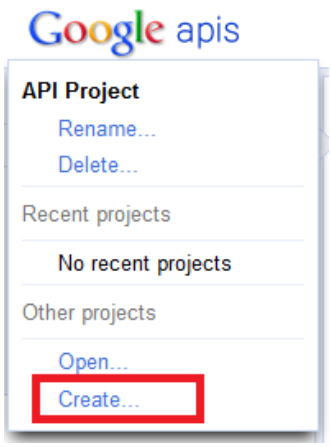

Figura D.2 Panel Google Apis

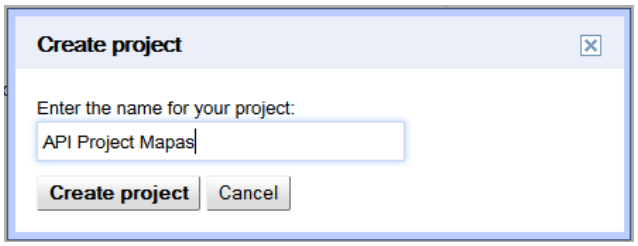

Figura D.3 Ventana para crear el proyecto

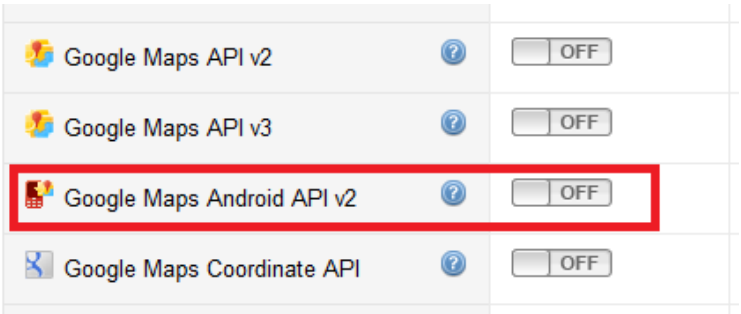

Figura D.4 Activación del API Key

Una vez activado aparecerá una nueva opción en el menú izquierdo llamada API Access. Accediendo a dicha opción se tiene la posibilidad de obtener una nueva *API Key* que permita utilizar el servicio de mapas desde una aplicación particular. Para ello, en la pantalla que se muestra en la figura D.5 se ha de pulsar el botón Create new Android key... .

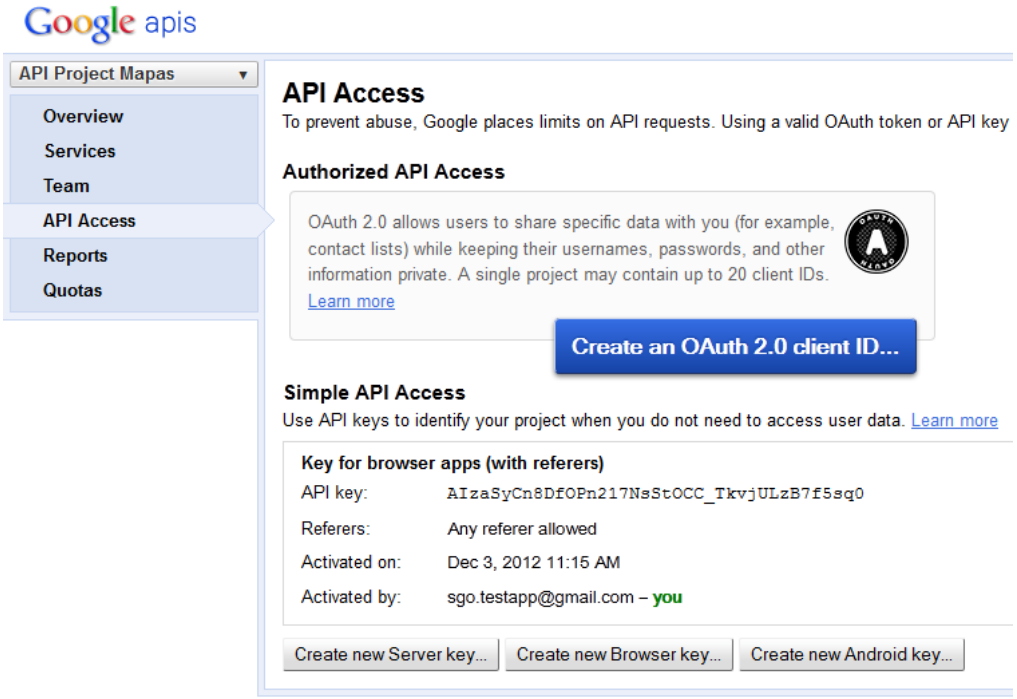

Figura D.5 Geneación de la API Key

Para crear la *API Key* es necesario introducir algunos datos identificativos de la aplicación en la pantalla de la figura D.7: la huella digital (SHA1) del certificado con el que se firma la aplicación, y el paquete java utilizado. Toda aplicación Android debe ir firmada para poder ejecutarse en un dispositivo, tanto físico como emulado. Este proceso de firma es uno de los pasos que se deben hacer antes de distribuir públicamente una aplicación. Adicionalmente, durante el desarrollo de la misma, para realizar pruebas y la depuración del código, se firma la aplicación con un "certificado de pruebas". Podemos saber en qué carpeta del sistema está almacenado este certificado accediendo desde Eclipse al menú Window > Preferences y se selecciona la sección Android > Build. Se muestra en la figura D.6

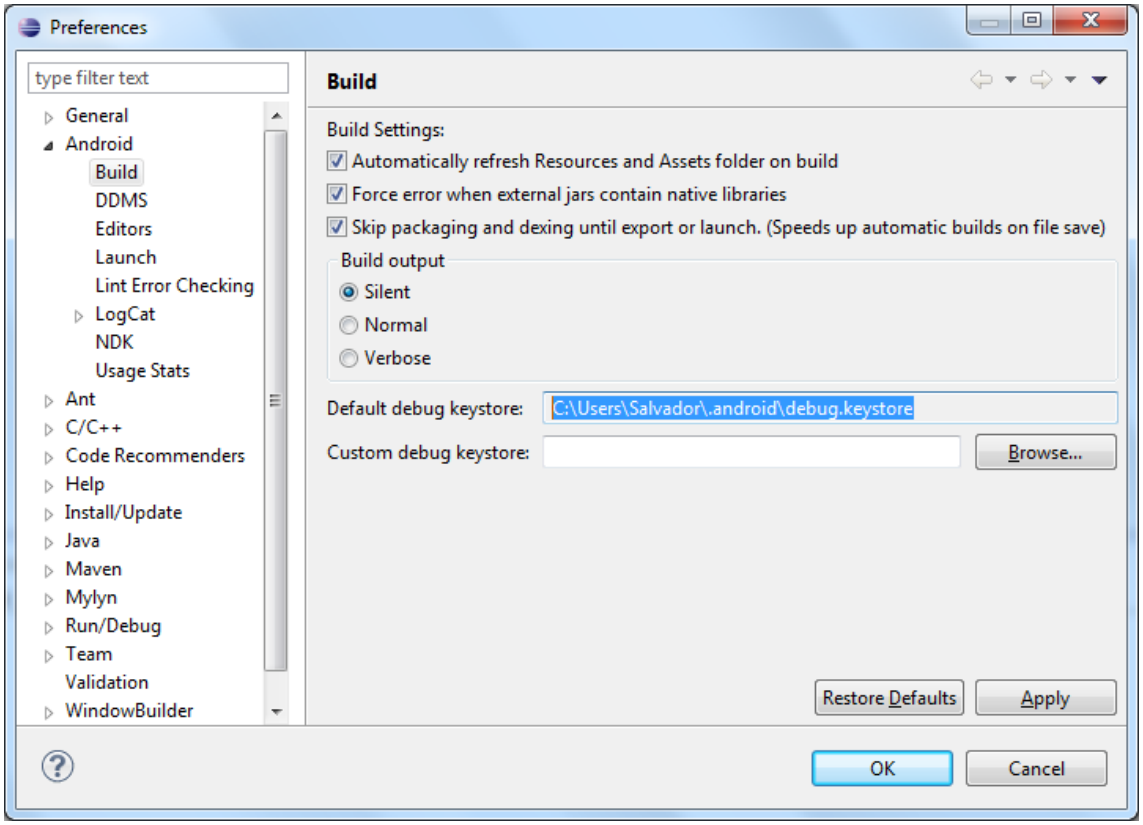

Figura D.6 Pantalla para ver la debug.keustore

Para obtener la huella digital SHA1 que corresponde al certificado de pruebas se debe acceder a dicha ruta desde la consola de comando de Windows y ejecutar los siguientes comandos:

```
C:\>cd C:\Users\Usuario\.android\

C:\Users\Usuario\.android>"C:\Ruta jdk\bin\keytool.exe" -list -v -
keystore debug.keystore -alias androiddebugkey -storepass android -
keypass android
```

Entre los datos que devuelve el comando se encuentra la huella SHA1. Con este dato se vuelve a la pantalla que se muestra en la figura D.7 y se copia en el cuadro de texto seguido de un punto y coma y del nombre del paquete java que se utilice en la aplicación. Al pulsar sobre el botón Create se genera la *API Key*. Se puede ver el valor de la clave en la pantalla siguiente dentro del apartado Key for Android Apps (with certificates) como se muestra en la figura D.8.

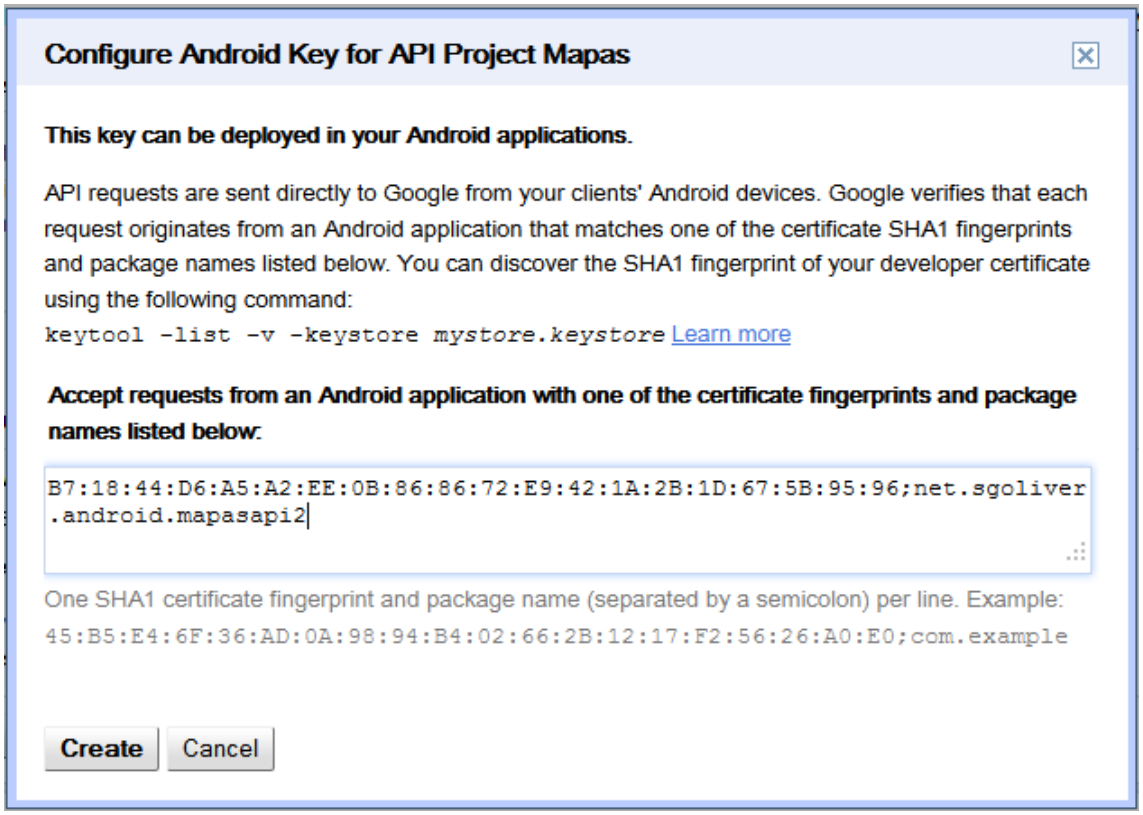

Figura D.7 Pantalla de introducción de los datos para generar la *API Key*

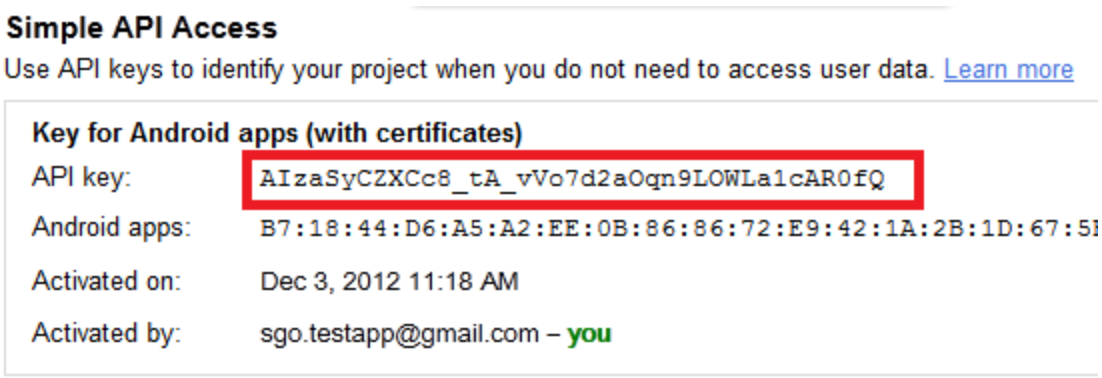

Figura D.8 *API Key generada*

Una vez se ha obtenido el valor de la API Key lo primero que se debe hacer es añadirla al fichero AndroidManifest.xml. Para ello se incluye en el fichero, dentro de la etiqueta <application>, un nuevo elemento <meta-data> con los siguientes datos:

```
...
<application>
...
    <meta-data android:name="com.google.android.maps.v2.API_KEY"
               android:value="api_key"/>
...
</application>
```

Además, se necesita añadir permisos adicionales para permitir acceder a los servicios web de Google, a Internet, y al almacenamiento externo del dispositivo (utilizado para la caché de los mapas):

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES
"/>
```

Por último, dado que la *API v2 de Google Maps* Android utiliza *OpenGL ES* versión 2, se debe especificar también dicho requisito en el AndroidManifest añadiendo un nuevo elemento `<uses-feature>`:

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

Una vez se ha configurado todo lo necesario en el AndroidManifest, se han de definir los elementos externos al proyecto. El primero de ellos será referenciar desde dentro del proyecto la librería con el SDK de *Google Play Services*. Para ello, desde Eclipse se importa la librería al conjunto de proyectos del espacio de trabajo mediante la opción de menú File > Import... > Existing Android Code Into Workspace. Esto nos lleva a la pantalla de la figura D.9, en la que se deberá seleccionar la ruta correcta. El resto de opciones se dejan con sus valores por defecto y se pulsa Finish para que Eclipse importe esta librería.

El siguiente paso será referenciarla desde el proyecto principal. Para ello se pulsa sobre botón derecho sobre el proyecto y se escoge la opción Properties que mostrará la pantalla de propiedades como se muestra en la figura D.10. Desde la esta ventana se accede a la sección Android, desde la que se puede añadir una nueva librería en la sección inferior llamada Library. Al pulsar el botón Add... aparece la librería recién importada, se selecciona y se añade a la lista de librerías referenciadas por el proyecto.

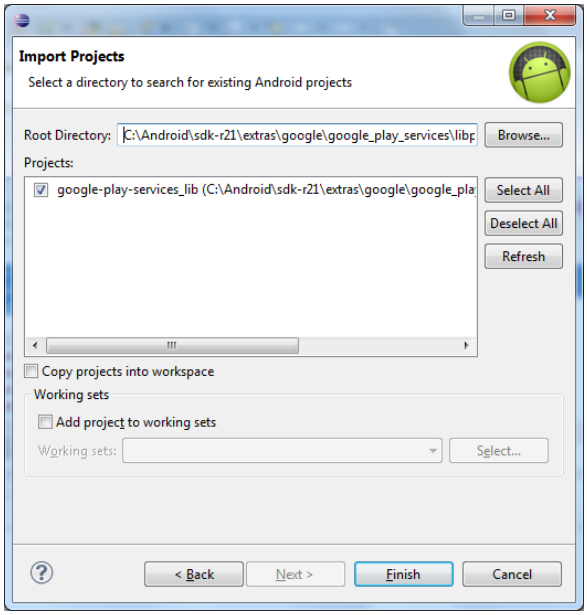

Figura D.9 Importación de la librería de Google Play

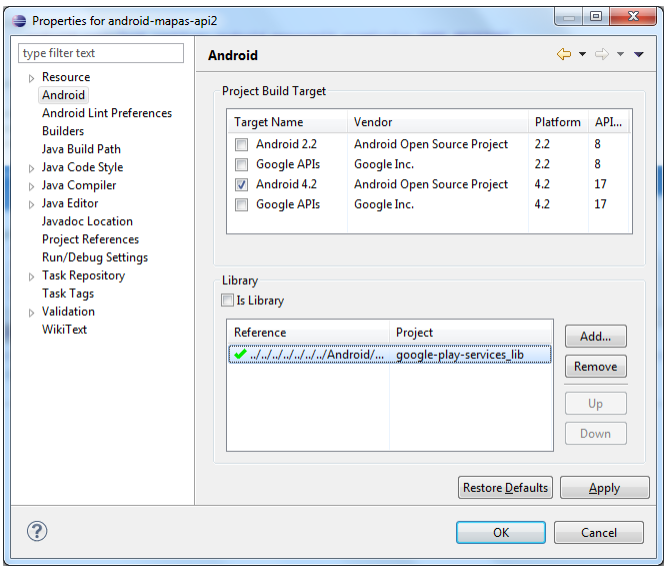

Figura D.10 Referenciación de la librería

Como último paso de configuración del proyecto, se puede incluir la librería de android-support-v4.jar, que permite que la compatibilidad de la aplicación con dispositivos que tengan instaladas versiones anteriores del sistema operativo (hasta la 2.2). Las versiones más recientes de ADT incluyen por defecto esta librería en los proyectos, pero si no lo está, se puede hacer mediante la opción del menú contextual Android Tools > Add Support Library... sobre el proyecto, o añadirla de forma manual como una librería corriente siguiendo los pasos del siguiente apartado.

D.5 Referencia a la librería de JSOUP

Para poder leer el código HTML de una página de Internet se ha de utilizar una librería externa. Para este proyecto se ha utilizado la librería JSOUP.

Lo primero es descargar el archivo .jar de la página: http://jsoup.org/download

Una vez descargado el fichero comprimido en el equipo se copia en el directorio del proyecto en una carpeta aparte (por convenio llamada lib). Si ya se ha añadido la librería de compatibilidad descrita en el paso anterior, la carpeta ya estará creada, si no, habría que crearla.

Una vez que ya se encuentra la carpeta en el directorio correspondiente, hace falta ligarla con el proyecto. Para ello se selecciona la pestaña Project > Properties para mostrar la ventana de propiedades como se ve en la figura D.11. En esta ventana se accede a la sección de Built Path y a la pestaña de Libraries. Dentro de esta pestaña se pulsa sobre el botón de Add jars que abre un navegador de archivos para seleccionar el archivo *jar* descargado.

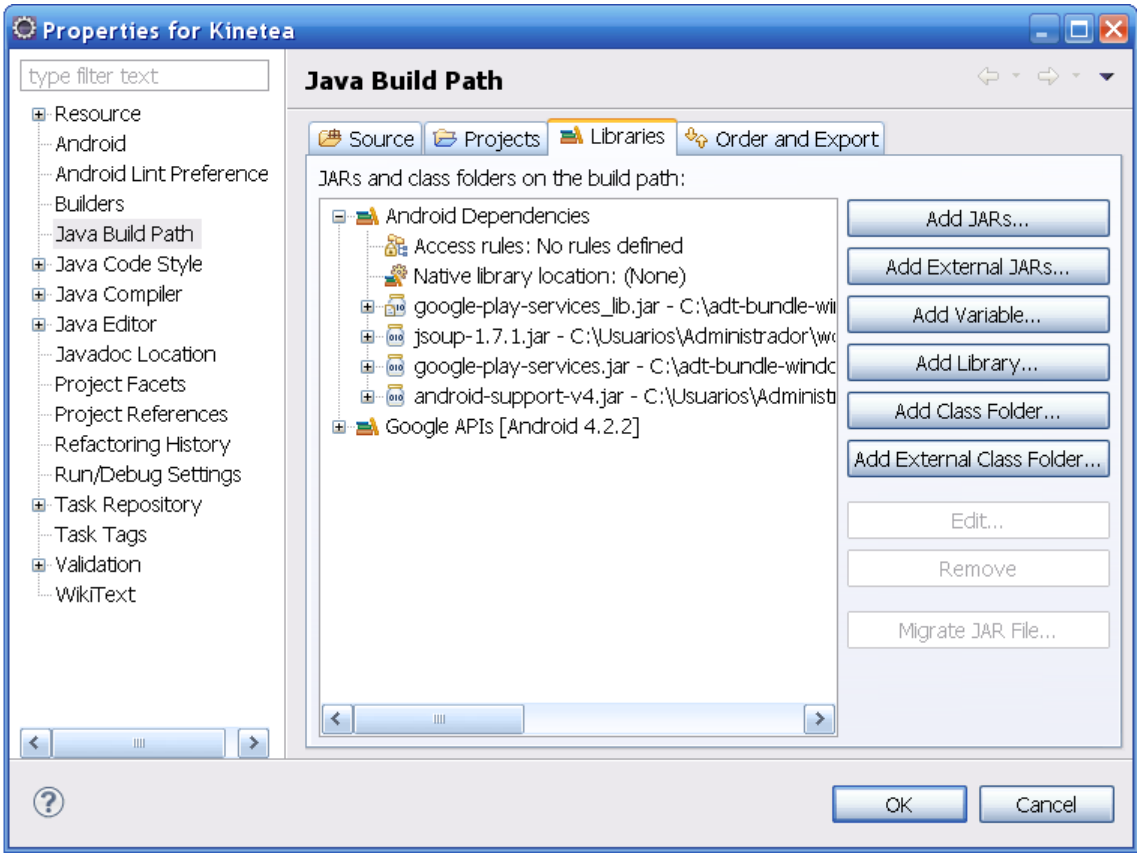

Figura D.11 Ventana para añadir *jar*'s externos al proyecto

D.6 Emulador Android

El Android SDK incluye un emulador virtual de un dispositivo móvil que se ejecuta en el equipo. El emulador permite crear prototipos, desarrollar y probar aplicaciones Android sin necesidad de utilizar un dispositivo físico.

El emulador simula todas las propiedades de *hardware* y *software* de un terminal móvil excepto la posibilidad de realizar llamadas. Proporciona una pantalla táctil y un teclado para simular los diferentes eventos que se pueden generar en un dispositivo Android real. El aspecto que tiene se puede observar en la figura D.12 [69].

Figura D.12 Pantalla del AVD

Una vez instalada la aplicación en el emulador se puede invocar a otras aplicaciones, conectarse a la red, reproducir audio y vídeo, almacenar y recuperar datos, crear notificaciones a usuario, etc. El emulador permite además varias capacidades para depurar la aplicación, como crear trazas en la aplicación para seguir la ejecución del código, simular una interrupción de la aplicación (como por ejemplo una llamada de teléfono) o simular caídas o efectos de latencia en los datos recuperados de Internet.

La configuración del AVD se realiza a través del AVD Manager incluido en el entorno de trabajo. Se pueden crear tantos AVD's como se desee y así probar la aplicación en diferentes terminales. Para cada uno se especifican la versión del sistema operativo, las características hardware, la capacidad de la tarjeta de memoria, etc. Para abrir el AVD Manager se puede hacer desde Eclipse o yendo a la carpeta de tools y ejecutando el archivo avd.exe. Se abrirá una pantalla como la mostrada en la figura D.13. Desde esta pantalla es posible añadir, modificar o borrar los AVD's

Para crear un nuevo AVD se debe pulsar en el botón New en la ventana de la figura D.13. Esto hará que se abra una ventana como la de la figura D.14 en la que se deben introducir las características hardware del emulador a crear.

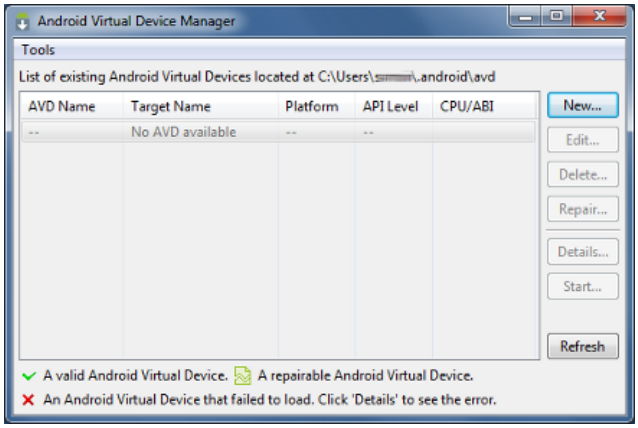

Figura D.13 Pantalla de creación del AVD

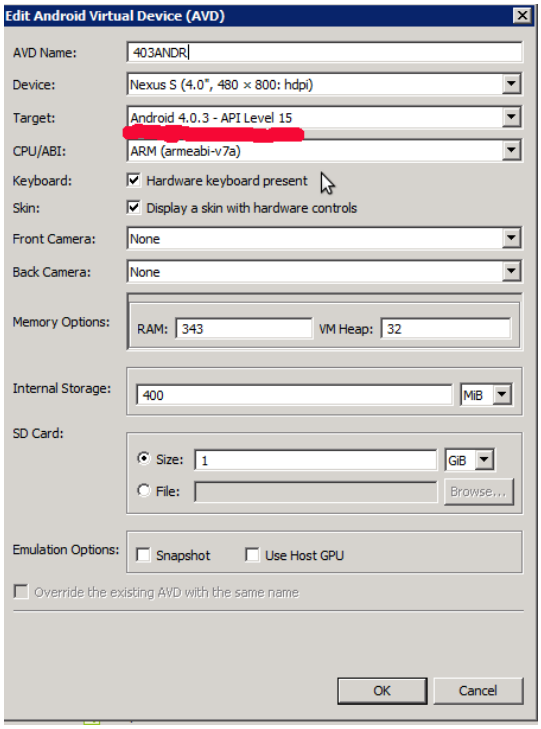

Figura C.14 Pantalla de configuración del AVD

Inicialmente, la nueva versión de mapas de Google maps para Android no es compatible con el emulador y se necesita de un dispositivo físico para poder hacer las pruebas. En vista a esto, la comunidad de desarrolladores de Android ha conseguido utilizar el emulador partiendo de una configuración determinada e instalando aplicaciones que actúan de soporte [70].

La configuración para crear los mapas es la que se muestra en la figura D.14. El dispositivo debe ser un Nexus S, la versión de Android la 4.0.3 – API Level 5 y se debe de incluir una memoria *RAM* y una capacidad de almacenamiento alta.

Una vez creado el AVD se deben de instalar las aplicaciones auxiliares sobre él, para ello la forma más sencilla es lanzar el emulador que acabamos de crear y desde la línea de comandos instalar los siguientes paquetes que se muestran junto a la *url* de descarga:

- GoogleLoginService.apk
  url: https://www.dropbox.com/s/lfde3juixuy88rg/GoogleLoginService.apk
- GoogleServicesFramework.apk
  url: https://www.dropbox.com/s/9kurwyhbbuecaea/GoogleServicesFramework.apk
- Phonesky.apk
  url: https://www.dropbox.com/s/9x8924gtb52ksn6/Phonesky.apk

Para instalarlos en el emulador se ejecuta lo siguiente en la línea de comandos:

```
adb shell mount -o remount,yourAvdName -t yaffs2
  /dev/block/mtdblock0 /system
adb shell chmod 777 /system/app
adb push GoogleLoginService.apk /system/app/
adb push GoogleServicesFramework.apk /system/app/
```

```
adb push Phonesky.apk /system/app/
```

Además de estos paquetes se deben de instalar los paquetes de Google Play services y de Google maps disponibles en las direcciones:

- Google Maps v.6.14.1
  url: https://www.dropbox.com/s/koo4wiwqg8agy8n/com.google.android.apps.maps-1.apk
- Google Play Services v.2.0.10
  url: https://www.dropbox.com/s/bh058hbrelccfsr/com.google.android.gms-2.apk

Y se instalan con los siguientes comandos:

```
adb install com.google.android.apps.maps-1.apk
adb install com.google.android.gms-2.apk
```

Combinando la configuración indicada y con estas aplicaciones adicionales ya es posible utilizar el AVD para depurar aplicaciones con la versión v2 de Google Maps para Android.

Anexo E. Manual de usuario

E.1 Inicio

En el momento en el que se abre la aplicación se lanza automáticamente la rutina de obtención de datos.

Si el GPS no está activado aparece una ventana de diálogo que permite acceder redirigir al usuario directamente a la zona de opciones en el que se habilita el GPS. Esta operación no se puede hacer de forma automática, por motivos de seguridad y privacidad se debe de activar manualmente. La activación del GPS no es obligatoria, ya que la aplicación puede obtener la ubicación mediante WPS o Cell-Id, aunque sí que se muestra como aconsejable.

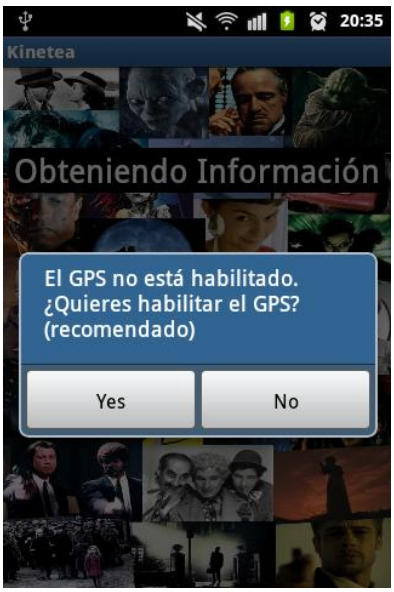

Figura E.1 Activación GPS

Una vez activos los sistemas de geolocalización disponibles, la aplicación escoge el que mejores prestaciones ofrece en la situación actual, y obtiene la ubicación del usuario. Con este dato se lanza la tarea en segundo plano que obtiene la información sobre los eventos. El proceso se divide en dos etapas, primero una de conexión a internet en la que se extraen los datos de la red y otra de procesamiento de datos. Durante la primera etapa, se muestra al usuario el texto “Conectando”, cuando finaliza esta y comienza la segunda se cambia el mensaje por “Obteniendo Información” acompañado de una barra de progreso que se va rellenando conforme se van procesando los datos.

Figura E.2 Conectando a la red

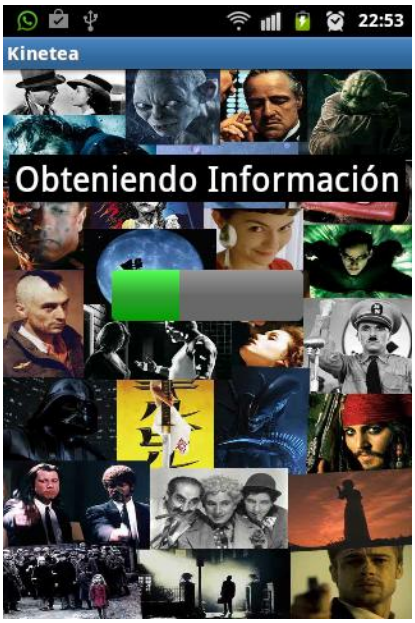

Figura E.3 Obteniendo información de la red

E.2 Pantalla Principal

Una vez que la tarea de obtención de datos ya se ha realizado se muestra al usuario la pantalla de inicio. La interfaz se compone de cuatro botones:

- Actualizar: Vuelve a calcular la posición del usuario y a lanzar la tarea de obtención de datos.
- Ver Eventos: Muestra un mapa en el que se colocan marcadores que representan la posición del usuario y la de los eventos.
- Ver Películas: Muestra una lista expandible en la que se muestran todas las películas disponibles en cartelera.
- Opciones: Despliega el menú contextual con las siguientes opciones:
    - Tipo de búsqueda: Permite seleccionar el tipo de eventos sobre los que se realizan las tareas de extracción de datos.
    - Info: Muestra una pantalla con información acerca de la aplicación
    - Ayuda: Muestra un pequeño manual de ayuda para el manejo de la aplicación.
    - Salir: Cierra la aplicación.

Figura E.4 Pantalla principal

E.3 Pantalla del Mapa

En esta pantalla se puede ver la situación del usuario y de los eventos sobre un mapa de Google. El mapa es totalmente interactivo y permite todas las opciones básicas, tales como acercar y alejar la cámara, girar la vista, cambiar el nivel de inclinación, mostrar los edificios en 3D, etc.

Se incluye un botón de opciones en la parte inferior del mapa que permite desplegar el menú contextual. Este menú contiene tres opciones:

- Elección del tipo de mapa: Se puede escoger entre una vista normal, desde satélite o híbrida.
- Elección del tipo de transporte: Permite escoger entre los tres medios de transporte disponibles para el cálculo de rutas: a pie, en coche o en transporte público.
- Ver detalles de ruta: Si ya se ha calculado una ruta se puede acceder a una pantalla en la que se listan las indicaciones para recorrer el trayecto paso a paso.

Para representar los cines y teatros en el mapa se utilizan marcadores situados en la posición del evento al que simbolizan. Los de color azul representan a un cine, los de color verde a un teatro y se incluye otro de color rojo que representa la posición del usuario. Al pulsar sobre un marcador aparece una ventana de información sobre el que se especifican características sobre el evento, como el nombre del cine o de la obra de teatro, dirección y/o teléfono del lugar y distancia al usuario. Si se pulsa sobre esta ventana de información aparece una ventana de diálogo, que ofrece al usuario mostrar información detallada sobre el evento u obtener la manera de llegar hasta él desde la posición actual.

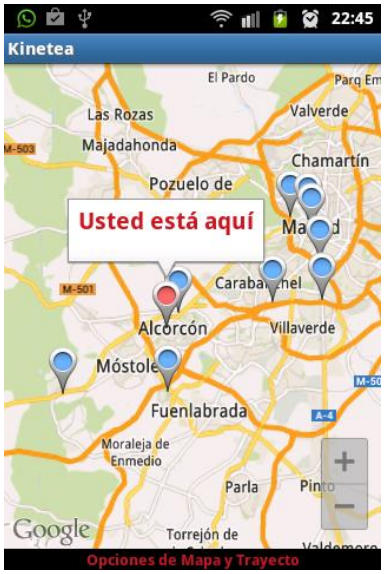

Figura E.5 Mapa posición usuario

Figura E.6 Mapa ventana cine

Figura E.7 Diálogo cine

Si se elige un evento de tipo teatro y se selecciona la opción de ver información, se lanza el navegador para redirigir al usuario a la página web que incluye las características detalladas de la obra y contiene el enlace que permite realizar la compra de la entrada.

Si se elige un evento de tipo cine, se ofrece la opción de ver su cartelera. La cartelera se muestra en una lista expandible en la que se muestran las películas ordenadas. Al pulsar sobre cualquier película se despliega la lista con los horarios de las sesiones disponibles. Al pulsar sobre un horario aparece una ventana de diálogo que permite redirigir al usuario a la página para comprar la entrada para la sesión elegida.

En esta pantalla de cartelera del cine seleccionado aparecen tres botones para facilitar la navegación a través de la aplicación. Así, se puede volver al mapa sin más, mostrar cómo llegar hasta el evento en el mapa o ir al menú principal.

Figura E.8 Lista de Cine

En ambos casos, al pulsar sobre un marcador, aparece, además, la opción de calcular la ruta desde la ubicación del usuario hasta la posición del evento seleccionado. Para el cálculo de la ruta se utiliza el medio de transporte que esté seleccionado en ese momento y que, como se ha visto antes, es modificable desde el menú de opciones del mapa. La ruta calculada se pinta en el mapa. Es posible calcular la ruta para varios medios de transporte y pintarlas en el mapa simultáneamente, pero se ocultan si se selecciona otro evento.

Al calcular la ruta también se obtienen las indicaciones para recorrerla paso a paso. Estas indicaciones se muestran al seleccionar el botón de "Ver Ruta Detallada" del menú contextual de la pantalla del mapa. Los detalles de la ruta se muestran sobre una lista en la que se indica el origen, destino y los pasos intermedios entre éstos. Desde la misma pantalla se muestran las pestañas de los tres tipos de transporte. Para visualizar los detalles de la ruta para otro medio de transporte se pulsa sobre la pestaña correspondiente, si ya se ha calculado la ruta, se muestran, si no, se pregunta al usuario si desea calcularla.

Figura E.9 Trayecto

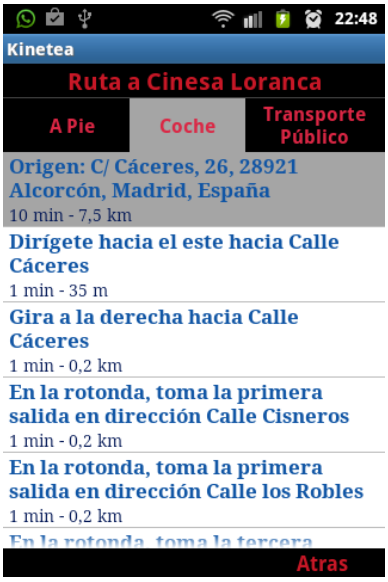

Figura E.10 Detalles trayecto

E.4 Lista de Películas

Como se ha dicho anteriormente, se pude visualizar la cartelera a partir de las películas en lugar de hacerlo por cines en el mapa. En el caso en que el usuario realice una búsqueda únicamente por teatros, el botón que lanza esta funcionalidad no estará disponible en la pantalla principal.

Las películas se listan ordenadas por géneros siguiendo las preferencias del usuario. Cada género tiene un ranking que aumenta cada vez que se accede a la compra de una entrada. De esta manera se mostrarán primero las películas que tienen una mayor probabilidad de ser escogidas. Dentro de cada género, se ordenan dependiendo de su importancia, dando preferencia a los estrenos y películas más taquilleras.

En cada elemento de la lista se incluye, junto al título e información de ésta, un botón que permite acceder a un mapa con la posición de los cines en los que se proyecta dicha película en relación a la posición de usuario. Este mapa es similar al que se ha utilizado anteriormente, también permite el cálculo de rutas y la visualización de los detalles del trayecto.

Al pulsar sobre cualquiera de las películas se despliega otra lista que muestra todos los cines en los que se proyecta, esta segunda lista está ordenada por proximidad. Al pulsar sobre cualquiera de estos cines se desplegará otra lista que muestra el horario de las sesiones de la película en ese cine. Pulsando sobre cualquier horario aparecerá una ventana de diálogo, como ocurría cuando se visualizaba la cartelera de un cine, permitiendo al usuario dirigirse a la página correspondiente para realizar la compra de la entrada.

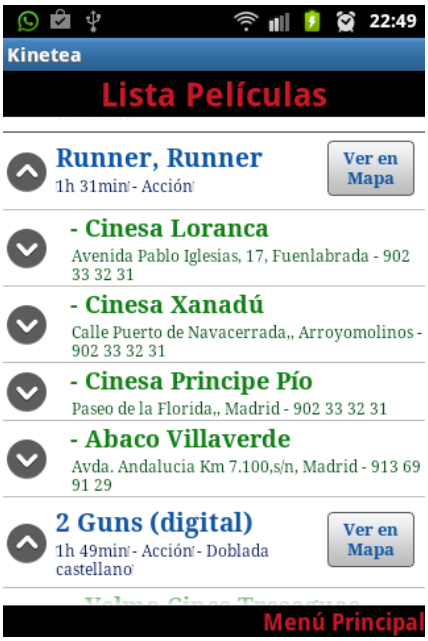

Figura E.11 Lista Peliculas

Figura E.12 Mapa películas

E.5 Eventos Cercanos

En el momento en el que se posiciona al usuario y se produce la obtención de los datos, bien sea al arrancar la aplicación o al actualizarlos, se comprueba si el usuario se encuentra cerca de la entrada de alguno de los eventos encontrados. En el caso de que esto ocurra, en lugar de mostrar la pantalla de inicio, aparecerá directamente la correspondiente al evento. Si es un teatro, la pantalla que se mostrará será un navegador con la página correspondiente a la información de la obra. Si es un cine, se abrirá la lista con su cartelera. Desde cualquiera de estas dos pantallas se puede volver al menú principal y navegar por la aplicación de la forma usual.

Es posible que dos o más eventos se encuentren muy cerca uno del otro y que la aplicación no sepa con certeza cuál pretende entrar el usuario y no pueda decidir cuál mostrar. En este caso se muestra una pantalla con una lista en la que se encuentran todos los eventos que cumplan la condición para que el usuario pueda acceder a las características del que elija. De nuevo, desde cualquiera de estas pantallas se puede volver al menú principal para realizar la navegación por la aplicación normalmente.

Figura E.13 Eventos cercanos