Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques

Universitat de Barcelona

Reproductor Musical Android amb geolocalització d'esdeveniments musicals

---

Víctor Montolío Mollá

Director: Simone Balocco

Realitzat en: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 17 de gener de 2013

Contingut

Taula d'Imatges

1. Introducció i antecedents

1.1 Introducció

En els últims anys el món dels Smartphones ha tingut un gran creixement exponencial que ha permès el llançament de milions d'aplicacions en les diferents plataformes de desenvolupament. Android ha sigut una de les que ha tingut un gran creixement.

Android no es la única plataforma que existeix dins d'aquest món, ha de competir amb grans marques en el món de la tecnologia. Dos d'aquestes grans marques son Apple i Microsoft i en un menor terme Blackberry. El gran avantatge que ha tingut Android ha sigut la gran quantitat d'aplicacions gratuïtes que Apple o Microsoft no tenen, un exemple seria la aplicació WhatsApp, una de les aplicacions mes descarregades arreu del món, on la podem descarregar gratuïtament des de Android mentre que per Iphone es de pagament.

En el següent paràgraf s'explicarà una mica més sobre la plataforma d'Android ja que es la que ha estat escollida per la realització del PFG.

1.2 Android

Android es una plataforma software i un sistema operatiu per a dispositius mòbils basat en el nucli de Linux. Va ser creat per Google i la Open Hanset Alliance. Pero Android es molt mes que un sistema operatiu, es també un llenguatge de programació i un framework per el desenvolupament d'aplicacions. Es a dir, el conjunt de tots aquets elements s'anomena Android. En el projecte col·laboren milions de persones arreu del mon, ja que es un

projecte de Software Lliure i qualsevol pot descarregar-se el codi font des de la pàgina d'Android. Igual que es fa amb els sistemes operatius GNU/Linux d'escriptori.

1.2.1 Història d'Android

Al mes de juliol de 2005, Google va comprar una petita empresa de Califòrnia anomenada Android Inc amb la intenció de desenvolupar un sistema operatiu mòbil que pogués respondre a les expectatives dels dispositius mòbils que començava a enlairar.

D'aquesta manera es va començar a treballar per que, al novembre de 2007, s'anunciés oficialment la creació de la Open Handset Alliance que vindria acompanyat pel llançament oficial del Android Software Development Kit(SDK). Aquesta primera versió va suposar un punt de partida com una forma d'avaluar la acceptació del mercat, per el que fins a mitjans d'agost del 2008 no va aparèixer la segona versió, coneguda com Android 0.9 SDK. Després de les comprovacions oportunes, a finals del mes de Setembre es va llençar oficialment la versió Android 1.0 SDK.

Encara que els inicis van ser una mica lents, degut a que es va llençar abans el sistema operatiu que el primer dispositiu, ràpidament es va col·locar com el sistema operatiu per mòbils mes venut del mon, situació que va arribar l'últim trimestre del 2010.

Al febrer de 2011es va anunciar la versió 3.0 d'Android, que estava optimitzat per tauletes en lloc de telèfons mòbils. Per tant Android ha passat de dispositius mòbils a dispositius mes grans.

1.2.2 Cota de mercat

Basant-nos en les estadístiques ofertes per StatCounter al Novembre de 2012, Android tenia una cota de mercat del 68,3% a Espanya, Figura 1.1. Sent així el sistema operatiu mòbil mes popular del país, seguit per iOS(29,7%) i SymbianOS(1,99%).

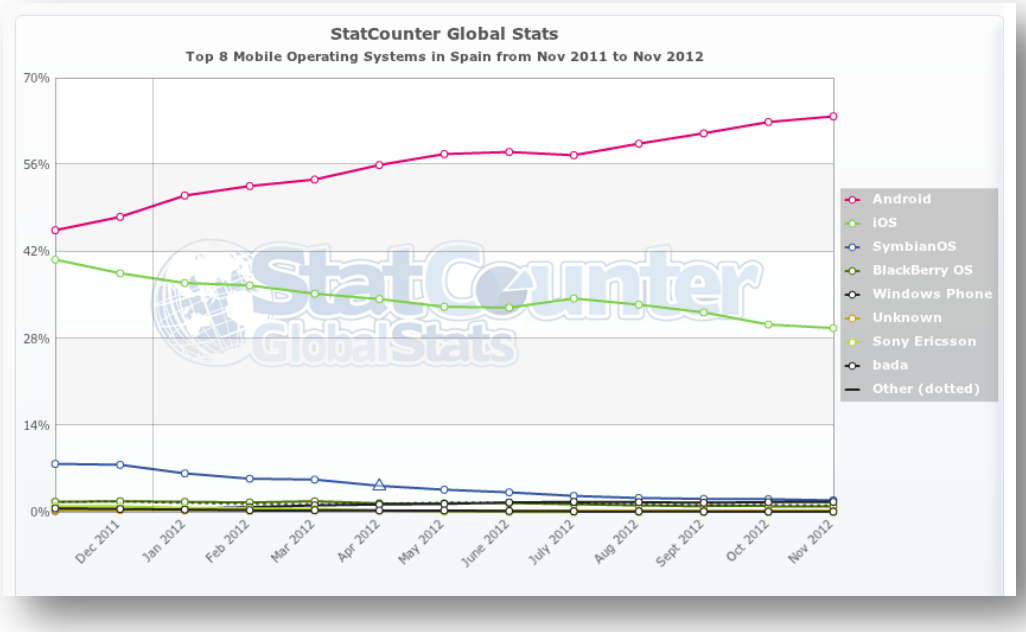


Figura 1 Cota de mercat a Espanya

Andorid també ha sigut globalment el numero 1 en la venta de dispositius mòbils, amb les estadístiques mostrades a la figura 1.2 on es pot comprovar aquest creixement exponencial.

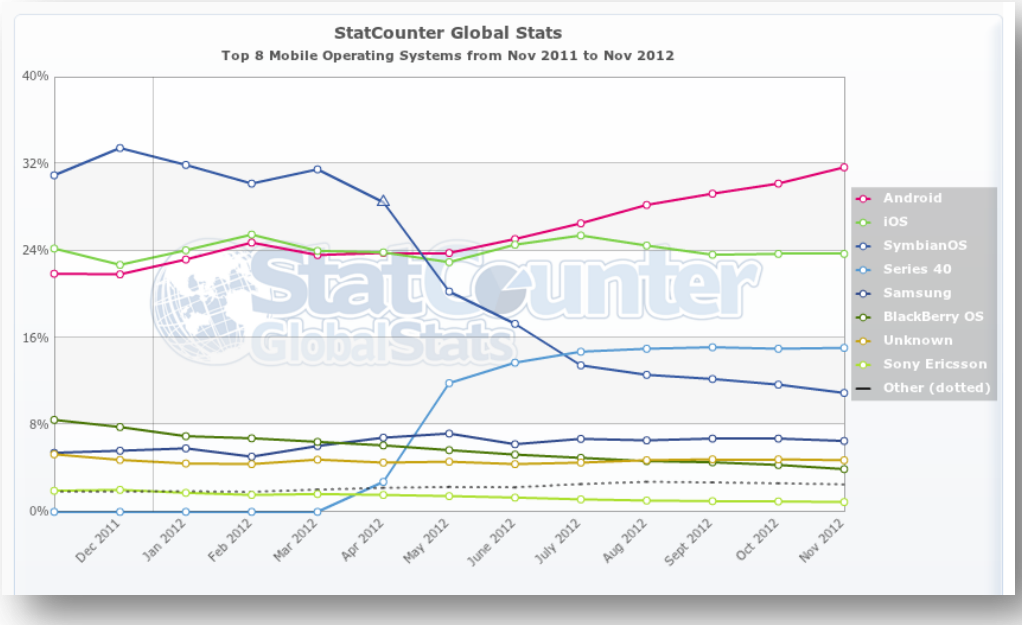


Figura 2: Cota de mercat al món

En quant a les estadístiques del us de les versions d'Android, basant-nos en les dades mostrades a la pàgina oficial d'Android sobre l'activitat abans del 3 de Desembre de 2012, podem trobar que la versió mes utilitzada es la versió Android 2.3.3+ (Figura 1.3). La versió més nova 4.x es un pas enrere ja que només els dispositius mes nous poden utilitzar-la.

Version	Codename	API	Distribution
1.5	Cupcake	3	0.1%
1.6	Donut	4	0.3%
2.1	Eclair	7	2.7%
2.2	Froyo	8	10.3%
2.3-2.3.2	Gingebread	9	0.2%
2.3.3-2.3.7		10	50.6%
3.1	Honeycomb	12	0.4%
3.2		13	1.2%
4.0.3-4.0.4	Ice cream sanwitch	15	27.5%
4.1	Jelly Bean	16	5.9%
4.2		17	0.8%

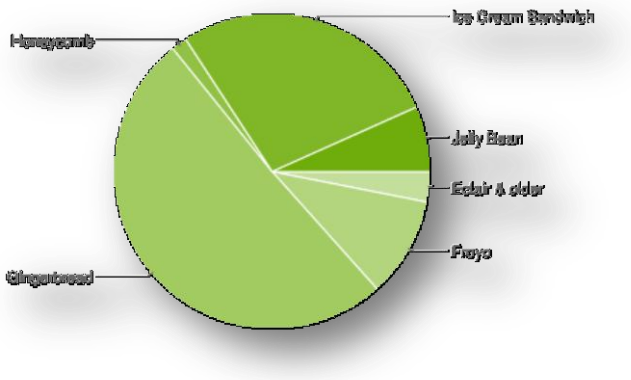


Figura 3: Versions d'Android

2. Definició d'objectius i motivació del problema.

2.1 Motivació del Problema

El món del desenvolupament per smartphones m'ha cridat l'atenció aquests últims anys arran de que ha sigut quan ha començat el seu gran creixement exponencial. A la carrera només vaig poder realitzar una pràctica sobre programació mòbil a l'assignatura de Nous usos per la informàtica ja que en el meu cas soc del pla d'adaptació i no vaig tenir l'opció de cursar l'assignatura que es realitza ara sobre mòbils. En aquest cas la motivació principal del PFG ha sigut aprofundir molt més en coneixements sobre Android per poder tenir bons coneixements de cara al món laboral.

Per una altra banda el mercat dels smartphones està en constant expansió i ofereix moltes possibilitats. Sobretot Android que va ser el gran revolucionari d'aquest gran món. Sent aquest un sistema de software lliure, codi obert i compatible amb gran part de dispositius, facilitant el desenvolupament d'aplicacions de terceres persones. Per aquest principal motiu Android s'està convertint en el sistema operatiu amb major quota de mercat .

2.2 Definició D'objectius

El Objectiu de desenvolupament del PFG es la creació d'una aplicació Android capaç de reunir alguna de les moltes possibilitats que ens ofereix un dispositiu Android. L'objectiu final serà la de tenir instal·lat en un dispositiu Android un reproductor musical capaç de realitzar consultes a Internet per tal que ens mostri la informació més important dels artistes de la nostra targeta SD i sigui capaç de localitzar-nos en un mapa la nostra posició actual i els esdeveniments del artista, per tal de que es pugui saber quin es el

esdeveniment més proper a nosaltres. L'objectiu final si es pogués seria el llançament de l'aplicació al Google Play per que els usuaris d'Android puguin descarregar-la i compartir-la.

2.3 Planificació del PFG

Abans de començar la realització del projecte final es va realitzar una previsió prèvia de les tasques que es tindrien que realitzar i el temps estimat per cadascuna d'elles. La divisió realitzada està representada en el següent diagrama de Gantt:

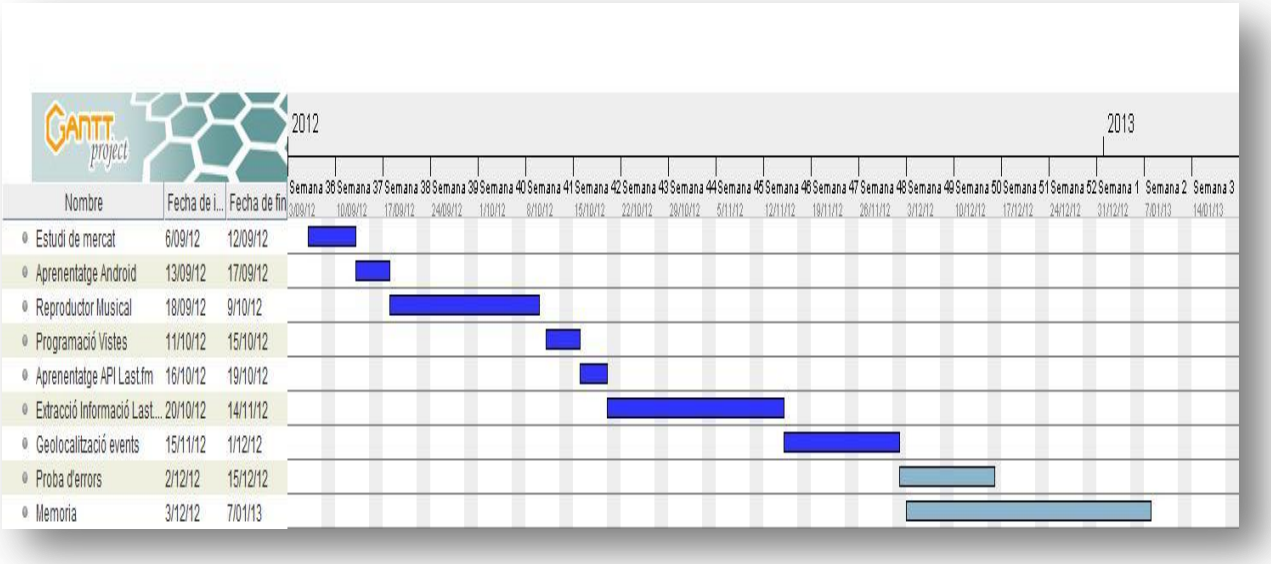


Figura 4: Previsió del temps de realització del PFG.

En la divisió de tasques es van tenir en compte els 7 dies de la setmana i que es podria realitzar feina tots els dies. Es va decidir que els dies entre setmana es realitzaria una mitjana de 6 hores, mentre que els caps de setmana el temps que es pogués. La

planificació de l'assignatura indica que son necessàries unes 450 hores per tal de realitzar correctament el PFG.

En el diagrama anterior es va planificar que per unes certes tasques es requeriria mes temps que per altres ja que es pensava que serien d'una complexitat mes elevada.

Una vegada acabada l'aplicació es pot dir que la panificació s'ha correspost bastant amb la planificació, però hem de tenir en compte que es difícil distingir les hores gastades en el aprenentatge i les hores gastades en la implementació, ja que aquestes dues tasques s'han pogut fer en paral·lel en la majoria de tasques. En la figura següent es mostra la planificació real del PFG i es pot comprovar com també s'ha arribat a afegir una nova tasca que es pensava en un primer moment que no es podria realitzar.

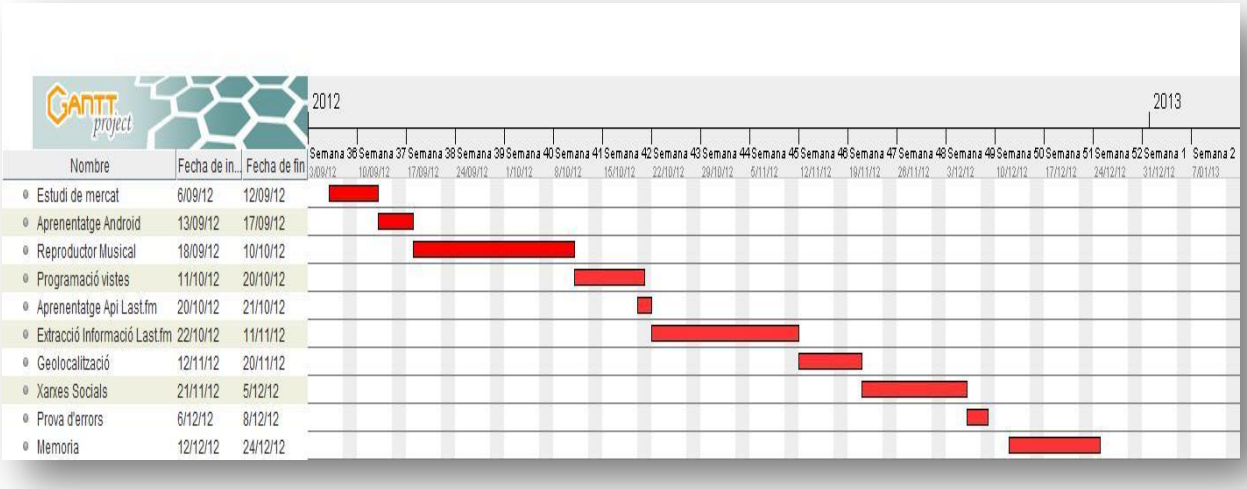


Figura 5: Temps real de realització del PFG

3. Pasos previs al desenvolupament.

L'aplicació escollida per la realització del projecte ha sigut un reproductor musical que sigui capaç de utilitzar alguns dels recursos dels que disposem al dispositiu Android. Utilitzarem la connexió a la xarxa per poder buscar informació des de la pàgina de Last.fm, s'utilitzarà SQLite per crear una petita base de dades per guardar la informació dels artistes, utilitzarem la localització per poder situar els esdeveniments musicals propers a la nostra posició i utilitzarem la possibilitat de connectar-nos a la xarxa social Twitter.

3.1 Anàlisi d'aplicacions del mercat

El mercat de la tecnologia mòbil està en constant evolució i per tant cada dia surten aplicacions noves i diferents que es podrien semblar a la proposta que es vol realitzar en aquest PFG, per tant s'ha de realitzar un anàlisi de les diferents aplicacions musicals que es troben actualment en el mercat d'Android.

El primer pas del desenvolupament del projecte va ser la realització d'un anàlisi del mercat actual d'Android en el tema d'aplicacions musicals. En aquest anàlisi vaig poder trobar que existeixen moltes aplicacions musicals però moltes d'aquestes o només buscaven informació per Internet sense l'opció de poder escoltar la teva o només permetien escoltar música. També vaig poder trobar diverses aplicacions que donaven la opció de escoltar música per Streaming o fragments d'una cançó i que després et dona l'opció de comprar la cançó sencera.

La següent taula es mostra una sèrie d'aplicacions amb les opcions descrites anteriorment.

Gigbeat	- Analitza la teva llibreria de música per trobar artistes. - Recerca centrada en la teva ubicació actual per mostrar-te només els Esdeveniments propers. - Fer el seguiment de múltiples ciutats i locals. - Notificació de concerts pròxims i de noves dates de gires. - Enllaç de compra d'entrades via Songkick - Sincronitza els teus artistes, concerts i locals favorits amb Songkick * - Sincronitza els teus artistes favorits con Last.fm ** - Esdeveniments d'artistes recomanats per Last.fm ** - Sincronitza els artistes amb Rdio *** - Fer "check in" a concerts en Foursquare **** - Nou widget per veure els pròxims concerts. * Necessita una conta en Songkick ** Necessita una conta en Last.fm *** Necessita una conta en Rdio **** Necessita una conta en Foursquar
Nvivo.es	-Escaneja la música que escoltes en el teu dispositiu mòbil per poder seguir els concerts de els teus artistes preferits fàcilment. -Alertes de concerts. Rep al teu mòbil una alerta quan el teu artista favorit realitzi un concert a la teva ciutat. -Consulta els concerts de la teva ciutat o de las ciutats que vulguis. -Comparador d'entrades. Afegim informació de més de 100 venedors.

	per a que la compris al millor preu. -Consulta les gires mundials dels teus artistes preferits. -Et recomanem concerts d'altres artistes basats en la música que escoltes. -Gestiona el teu perfil. Personalitza els teus tus artistes, sales i concerts preferits. -Podràs compartir els concerts a través de Twitter, Facebook i email.
Giger	-Importi els artistes preferits de la seva conta de last.fm -Els artistes de importació de la memòria del dispositiu (intern i SD) -Cerca de concerts basats en la seva ubicació, els artistes preferits. -Cóm arribar a la ubicació de concerts -Aconseguir una relació directa amb la compra de entrades a través de Songkick.com -Afegir esdeveniments al teu calendari. -Compartir informació sobre el concerts amb els teus amics. -Seguiment dels llocs preferits per els pròxims esdeveniments.
andEvents	Troba concerts, informació sobre artistes i música en el món. Troba esdeveniments buscant per artista, posició o sala de concerts.
Concierto tinta	-Recerca: Buscar concerts, de la teva ciutat en qualsevol part del món. -UBICACIÓ: Tinta Gigabit troba automàticament la teva ubicació. -FILTER: Filtrar per data. -Detall del concert: Faci clic en un esdeveniment pe veure detalls del esdeveniment, inclou el temps pe la data del esdeveniment. -CERCA: Detalls del concert.

eventseekr	- Milions de esdeveniments per tot el món que estan organitzat pe artista, categoria, data i lloc. - La capacitat de mantenir el seguiment de tots els seus artistes preferits. - Compartir esdeveniments amb amics a Facebook. - Un mapa amb els restaurants, hotels i bars propers. - Les dades per cada lloc. - La informació sobre els artistes i els seus esdeveniments propers. - La informació del lloc amb els pròxims esdeveniments.
Winamp	-Sincronització sense fils Gratuïta -Suport per sincronització amb Winamp per Mac (beta) -Importació en un clic de la biblioteca i les llistes de iTunes -Més de 50k estacions de radio en SHOUTcast. -Estacions SHOUTcast destacades. -Controls del Reproductor Persistents. -Noticies d'artistes, biografies, fotos i discografia.
RealPlayer	-Música, vídeos i fotos en un únic lloc -Disponible en nou idiomes -Gràfics optimitzats pe dispositius d'alta resolució -Llistes de reproducció -Scrobbling de last.fm -Comandes de veu per cerca -Control de reproducció amb el telèfon bloquejat. -Xarxes socials més populars per compartir fitxers. -Ajust d'una una cançó com a to de trucada.

	-Control d'auriculars
Spotify	- Accés a més de 15 milions de cançons. - Escolta en línia - Escolta en Modo Offline - sense connexió a Internet - Comparteix música amb els teus amics. - Destaca les teves cançons preferides - Passa per Wi-Fi els MP3 del teu ordinador al Android. - Crea i sincronitza llistes de reproducció. - Envia les cançons que escoltes directament a Last.fm i Facebook.

Una vegada hem donat un cop d'ull a les aplicacions del mercat que podrien ser semblants a la que es desenvoluparà en aquest PFG hem de destacar quines serien les opcions noves que aportaria la nostra aplicació. Moltes de les aplicacions que hem vist anteriorment disposen de moltes de les qualitats que es desenvoluparan, però la que es desenvoluparà en aquest PFG disposa de una barreja d'algunes de les mes importants i facilita la opció de connexió amb xarxes socials cosa que poques de les que s'han vist ho disposen. La qualitat principal seria la de buscar la informació dels artistes a la vegada que es pot escoltar la musica d'ells cosa que poques fan.

3.2 Introducció a la plataforma Android

El segon pas del desenvolupament va ser començar a practicar amb la tecnologia Android, ja que en el meu cas no havia fet molta cosa sobre aquest tema durant la carrera. Android té parts que son molt semblants a Java per tant només vaig tenir que endinsar-me molt més en lo que és la programació especifica en Android.

A continuació s'expliquen les característiques més importants de la plataforma de desenvolupament.

3.2.1 Característiques

Com s'ha explicat anteriorment Android es una plataforma software i un sistema operatiu per dispositius mòbils basat en un nucli de Linux.

Les característiques principals mes destacades del sistema son:

- **Connectivitat**
    - Navegador integrat, basat en el motor open source webkit.
    - Depenent del Hardware: GSM,Bluetooth, EDGE, 3G,WIFI.
- **Software**
    - Gràfics optimitzats amb OpenGL ES 2.0.
    - SQLite per emmagatzematge de dades estructurades(Base de dades).
    - Entorn de desenvolupament: emulador, Debugger.
- **Hardware**
    - Màquina virtual Dalvik: Base de trucades semblant a Java.

- Suport multimèdia amb els formats mes comuns d'àudio, vídeo e imatges.
- Depenent del telèfon: Càmera, GPS, Brúixola, acceleròmetre, pantalla tàctil.

3.2.2 Arquitectura

Abans de endinsar-nos en el desenvolupament d'aplicacions en Android es important conèixer com està estructurat el sistema operatiu. La estructura del sistema operatiu s'anomena arquitectura i en el cas d'Android està format per diverses capes que faciliten al desenvolupador la creació de les diverses aplicacions(Figura 3.1).

A continuació es fa una breu explicació de cada capa.

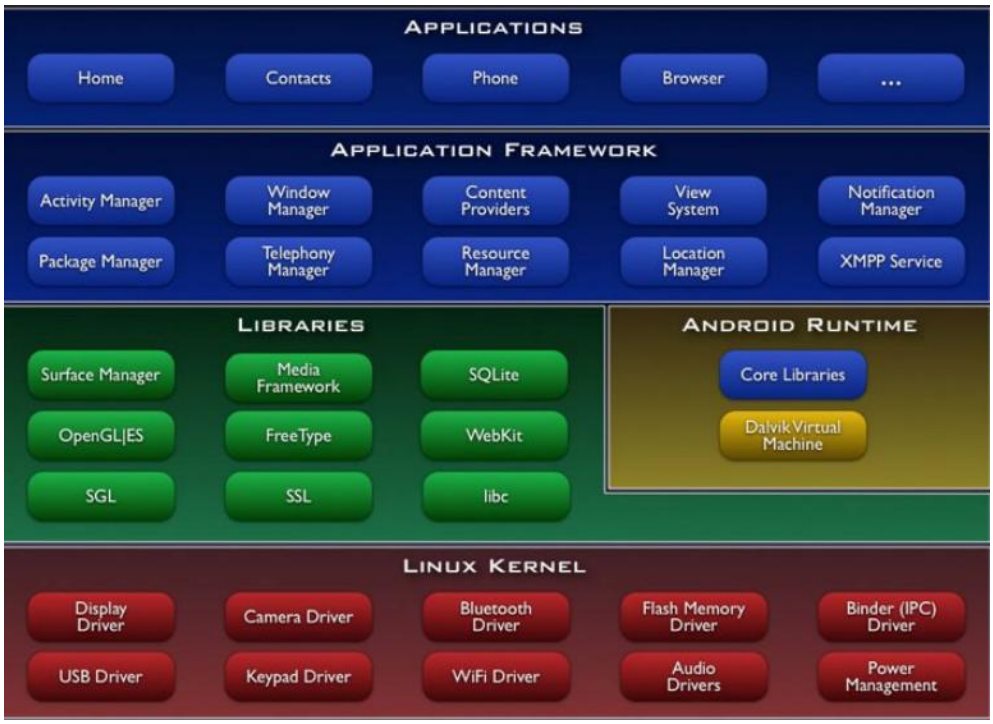


Figura 6 Arquitectura Android

1. **Kernel de Linux:**

   Es el nucli del sistema operatiu Android basat en el kernel de Linux de la versió 2.6, similar al que pot es pot incloure en qualsevol distribució de Linux, nomes que

aquest està adaptat a les característiques del hardware que executarà Android, es a dir per a dispositius mòbils.

El nucli actua com una capa d'abstracció entre el hardware i la resta de capes de l'arquitectura. El desenvolupador no accedeix directament a la capa, sinó que utilitza les llibreries disponibles de les capes superiors. D'aquesta forma també ens evitem el fet de conèixer les característiques de cada telèfon, per exemple si hem d'utilitzar la càmera el Sistema operatiu s'encarrega d'utilitzar la que té el telèfon. Per cada element existeix un driver dins del kernel.

El kernel també s'encarrega de gestionar els diferents recursos del telèfon i del sistema operatiu: processos, elements de comunicació, etc.

2. **Llibreries:**

Aquesta capa està situada just per sobre del kernel i està formada per les biblioteques natives d'Android, també anomenades llibreries. Estan escrites en llenguatge C o C++ i complides per l'arquitectura hardware especifica del telèfon. Aquestes normalment estan fetes pel fabricant, que també s'encarrega d'instal·lar-les al dispositiu. El objectiu principal de les llibreries es proporcionar funcionalitat a les aplicacions per tasques que es realitzen amb freqüència.

Entre les llibreries incloses habitualment trobem OpenGL(gràfics), Multimèdia(àudio, imatge i vídeo), Webkit(navegador), SSL(xifrat de comunicacions), FreeType(Fonts de text),SQLite(Base de dades) entre d'altres.

3. Entorn d'execució

El entorn d'execució en Android no està considerada una capa en si mateixa donat que també està formada per llibreries. Aquí trobem llibreries amb funcionalitats habituals a Java, així com altres especifiques d'Android.

El component principal del entorn d'execució d'Android es la màquina virtual Dalvik. Les aplicacions es codifiquen en Java i compilades en un format específic per que aquesta màquina virtual les pugui executar. El avantatge d'això es que les aplicacions es compilen una única vegada i d'aquesta forma estan preparades per distribuir-se amb la garantia de que es podran executar en qualsevol dispositiu Android que tingui la versió mínima del sistema operatiu.

S'ha de destacar que Dalvik es una variació de la màquina virtual de Java, per lo que no es compatible amb el bytecode de Java. Java únicament es el llenguatge de programació i els executables es generen amb el SDK d'Android i tenen la extensió .dex que es específic per a Dalvik, per aquest motiu no es pot executar aplicacions Java en Android ni a la inversa.

4. Framework d'aplicacions

Capa formada per totes les classes i serveis que utilitzen directament les aplicacions per realitzar les seves funcions. La majoria dels components d'aquesta capa son llibreries Java que accedeixen als recursos de les capes anteriors a partir de la màquina virtual Dalvik. En aquesta capa podem trobar:

- Activity Manager: Administra la pila d'activitats així com el cicle de vida.
- Windows Manager: Organitza el que es mostrarà a la pantalla. Crea les superfícies a la pantalla que posteriorment passaran a ser ocupades per les activitats.
- Content Provider: Crea una capa que encapsula les dades que es compartiran entre aplicacions per tenir el control de com s'accedeix a la informació.
- Views: Les vistes son els elements que ens ajuden a construir les interfícies d'usuari: botons, quadres de text, llistes i fins elements mes avançats com un navegador Web o un visor Google Maps.
- Notification Manager: Engloba els serveis per notificar al usuari quan alguna cosa requereixi la seva atenció mostrant alertes en la barra d'estat. Una dada important es que aquesta biblioteca també permet jugar amb sons, activar el vibrador o utilitzar els LEDs del telèfon.
- Package Manager: Aquesta biblioteca permet obtenir informació sobre els paquets instal·lats en el dispositiu Android, a part de gestionar la instal·lació de nous paquets. Amb paquet ens referim a la forma en la que es distribueixen les aplicacions Android, son fitxers .apk que a la vegada inclouen els fitxers .dex amb tots els recursos i fitxers addicionals que necessita l'aplicació.
- Telephony Manager: Llibreria encarregada de poder realitzar trucades o enviar i rebre SMS/MMS, encara que no pot substituir o eliminar l'activitat que es mostra quan una trucada està executant.
- Resource Manager: Amb aquesta llibreria podrem gestionar tots els elements que formen part de l'aplicació i que es troben fora del codi, es a dir, cadenes de text traduïdes, imatges, sons o layouts.
- Location Manager: Permet determinar la posició geogràfica del dispositiu Android mitjançant GPS o xarxes disponibles i treballar amb mapes.

- Sensor Manager: Permet manipular els elements de hardware del telèfon com el acceleròmetre, sensor de llum etc.
- Càmera: Podrem fer us de les càmeres del dispositiu per fer fotos o vídeos.
- Multimèdia: Permet reproduir i visualitzar àudio, vídeo e imatges en el dispositiu.

5. Aplicacions

Es la última capa que inclou totes les aplicacions del dispositiu, tant les que tenen interfície gràfica com les que no en tenen; les natives(programades en C o C++) i les administrades(programades en Java); les pre-instal·lades i les que el usuari instal·la.

També podem l'aplicació principal del sistema: Inici(Home) o llançadora(launcher), ja que es la que permet executar altres aplicacions mitjançant una llista, mostrant diferents escriptoris on es poden posar accessos directes a aplicacions o fins i tot widgets, que també aplicacions d'aquesta capa.

3.3 Programació en Android.

En aquest apartat podrem trobar diferents elements que seran necessaris conèixer a l'hora d'iniciar-se en la realització de l'aplicació Android.

Es començarà explicant els conceptes bàsics comunes a totes les aplicacions Android i a continuació mes en detall en l'aplicació realització i en la utilització de les diferents eines del dispositiu.

3.3.1 Conceptes bàsics de la programació en Android

A l'hora de fer una aplicació per Android s'han de tenir en compte una sèrie de conceptes per realitzar un bon desenvolupament, el mes important està descrit a continuació:

- **Activitat(Activity):** Component d'una aplicació que presenta una interfície gràfica al usuari. Tota interfície necessita una activitat.
- **Cicle de vida d'una activitat:** Al sistema, les activitats s'emmagatzemen en una pila. Quan una nova activitat es iniciada, es col·loca al principi de la pila i comença a executar-se. Les activitats anteriors no s'executaran fins que les que té a sobre es tanquin.

  El cicle de vida d'una activitat té principalment quatre estats:

  1. ***Execució(running):*** Si la activitat està en pantalla.
  2. ***Pausada(Paused):*** Si una altra activitat es troba per davant. Una activitat pausada està totalment viva, però pot ser eliminada pel sistema.

3. ***Parada(stoped):*** Si una altra activitat la tapa completament encara conserva els estats i la informació, però no es visible per l'usuari, per tant el que s'oculta a la finestra i pot ser eliminada si es necessita memòria.
4. ***Finalitzada(Finished):*** Si una activitat està en pausa o parada el sistema pot eliminar-la tancant el procés.

La figura següent (Figura 3.2) mostra els canvis d'estat mes importants. Els quadres rectangulars son funcions re-programables per ampliar les operacions realitzades quan es canvia d'estat. Els quadres pintats son els estats principals en els que pot estar una activitat.

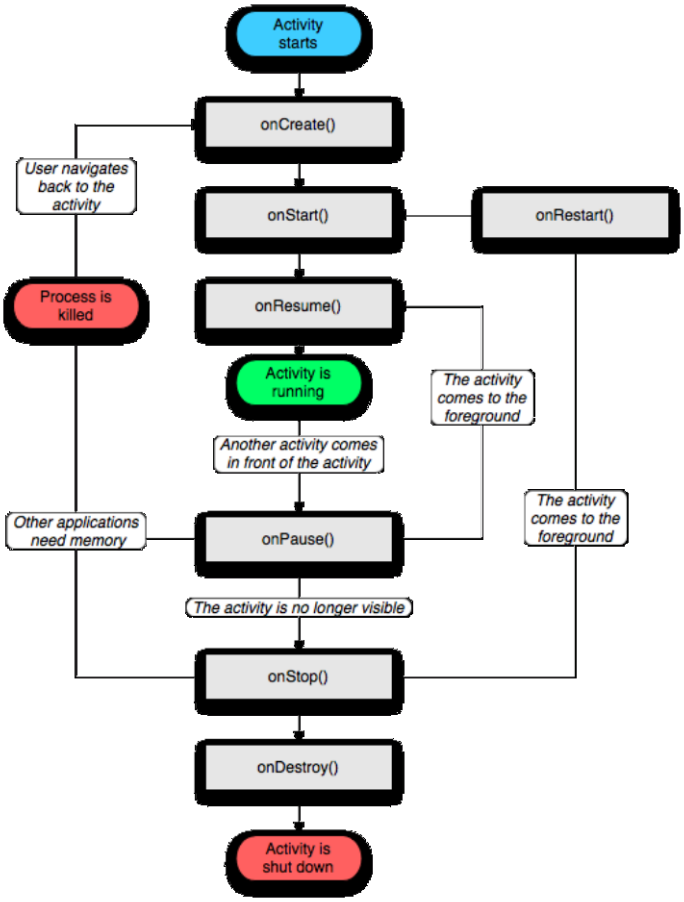


Figura 7: Cicle de vida d'una activity.

- **Services:** Comunament es coneix com a procés. Aquests seguiran executant-se encara que no hi hagi cap interfície gràfica. Per exemple, en el nostre cas quan

s'ha de minimitzar el reproductor musical la música seguirà sonant ja que s'haurà creat un servei.

- **Intents:** Mecanisme per poder comunicar diverses aplicacions i activitats. Poden notificar, per exemple, esdeveniments del sistema, com es connecta el cable USB. També poden ser programades per demanar a altres aplicacions que executin alguna acció.
- **Content Provider:** Proveeix dades a d'altres aplicacions(Proveïdor de SQLite).
- **Broadcast Receiver:** Component per la execució de tasques petites en segon pla. S'utilitza per a que una aplicació respongui a un determinat esdeveniment del sistema.

3.3.2 Interfícies Gràfiques.

Com s'ha explicat en el apartat tota interfície gràfica necessita una activitat per poder ser mostrada per pantalla, però per poder crear una interfície gràfica també es necessari un *layout*.

Com s'ha explicat en el apartat tota interfície gràfica necessita una activitat per poder ser mostrada per pantalla, però per poder crear una interfície gràfica també es necessari un *layout*.

- Declarar elements en XML: Android proporciona un senzill vocabulari que corresponen a les classes i subclasses de les vistes, així com pels widgets i els layouts.
- Inicialitza els elements en una rutina: Es poden crear diferents vistes i manipular les seves propietats en el codi de programació.

El Framework Android et dona la possibilitat d'utilitzar qualsevol dels dos mètodes o inclús els dos a la vegada. Per exemple es pot crear l'estructura principal de la pantalla en un XML i a la vegada modificar les propietats dels elements quan codifiquem el codi.

El avantatge de declarar la UI dins del codi XML es que et permet separar millor la presentació de la teva aplicació del codi que controla el funcionament. Amb això la descripció de la UI serà extern al codi de l'aplicació així es podrà modificar la interfície sense tenir que tocar el codi principal de l'aplicació.

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

Figura 8: Exemple de Layout XML

A continuació s'expliquen alguns dels dissenys mes comuns que es poden construir amb la plataforma Android.

- **Linear Layout:** Aquest layout organitza els seus fills en una sola fila horitzontal o vertical. Es crea una barra de desplaçament si la mida de la finestra excedeix la mida de la pantalla.

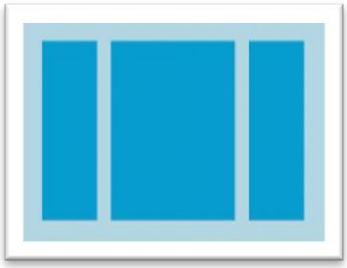


Figura 9: Linear Layout

- **Relative Layout:** Et permet especificar la localització dels objectes secundaris amb relació a un altre, fill A cap a l'esquerra del fill B, o amb relació al pare, alineat amb la part superior del pare.

Figura 10: Relative Layout

3.3.3 Fitxer AndroidManifest.xml

Una aplicació en Android no només té un sol fil d'execució, té diferents components que interactuen amb l'aplicació. Els components d'una aplicació Android son iniciats, o cridats per realitzar una determinada acció, mitjançant un Intent específic.

Per poder definir de quins components està formada tota la nostra aplicació, i que Intents poden interactuar amb la mateixa, s'utilitza el fitxer manifest. El sistema Android utilitza aquest fitxer per conèixer la nostra aplicació, així com quina es l'activitat principal.

El fitxer manifest no només defineix els components d'una aplicació. La següent llista resumeix les parts mes importants pel disseny de la nostra aplicació.

- La versió de l'aplicació, per poder realitzar actualitzacions al Android Market.
- Les versions d'Android en les que es pot utilitzar la nostra aplicació.
- Els requisits Hardware de l'aplicació.
- Permisos per utilitzar diferents components, com per exemple Internet.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="pfg.paquet.reproductor"
        android:versionCode="1"
        android:versionName="1.0" >
        <uses-sdk android:minSdkVersion="10" android:targetSdkVersion="8" />
        <uses-permission android:name="android.permission.WAKE_LOCK" />
        <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
        <uses-permission android:name="android.permission.INTERNET"/>
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
        <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-library android:name="com.google.api.translate"/>
```

Figura 11: Primer tros de AndroidManifest.xml

En el tros de codi anterior podem veure una primera part del fitxer manifest, els elements XML mes rellevants son:

- Element *<manifest>*: Es el element principal del fitxer, dins d'aquesta etiqueta definim els components de l'aplicació, permisos, requisits de hardware i versions suportades. Aquests son els atributs bàsics:
    - *versionCode i versionName*: defineixen la versió de la nostra aplicació. v*ersionCode* a d'anar incrementant-se cada vegada que publiquem una versió de l'aplicació i *versioName* es mostrarà als usuaris que la busquin.
- Element *<uses-permission>*: Android té un sistema de seguretat ben elaborat. Cada aplicació es executada en el seu propi procés i maquina virtual, amb el seu propi usuari Linux i Grup, i no pot interferir amb altres aplicacions. Es restringeix la

utilització de recursos del sistema, com accés a Internet, targeta SD,etc. Si la nostra aplicació vol utilitzar un d'aquests recursos hem de demanar permís. Aquest permís es demana amb aquest element. En el nostre cas demanem permís de connexió a Internet, llegir el estat del telèfon, accés al calendari.

- Element <*uses-sdk*>: Cada versió d'Android té un número associat, conegut com SDK version. Especifica la mínima versió suportada, com la mínima versió per la que va dirigida, els atributs son *minSdkVersion* i *targetSdkVersion*.

```
<application

	android:icon="@drawable/logoaplicacio"

	android:label="@string/app_name" >

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

			<activity android:name=".TabLayoutActivity"
		android:configChanges="orientation|keyboardHidden"

		android:label="@string/app_name" android:screenOrientation="portrait">

			<intent-filter>

				<action android:name="android.intent.action.MAIN" />

				<category android:name="android.intent.category.LAUNCHER" />

			</intent-filter>

		</activity>
```

Figura 12: Segon tros de AndroidManifest.xml

Els elements d'aquest segon tros de codi del fitxer manifest son:

- Element <*application*>: Una aplicació pot contar amb molts fitxers XML no només el fitxer *manifest*. Per poder referenciar a un atribut d'un altre fitxer s'utilitza "@" al principi del nom, seguit pel nom de la carpeta dins del directori */res/* i finalment el nom del atribut.

  - El atribut *icon* fa referencia a la imatge en */res/drawable* que s'utilitzarà com a icona principal de l'aplicació en els menús del dispositiu.
  - El atribut *label* es el nom que apareixerà sota la icona dels menús.
- Element <*activity*>: Definició de les activitats que l'aplicació utilitzarà. Els atributs son els següents:
  - El atribut *name* defineix la ruta a l'activitat principal del paquet especificat en el element <*manifest*>.
  - El atribut label serà mostrat en la barra de títol de l'activitat.
  - El atribut *screenOrientation* especifica l'orientació que utilitzarà l'activitat. Es important definir-lo per poder forçar una orientació concreta, en lloc de que canvii automàticament al girar el mòbil. Al assignar-li *portrait* definim orientació vertical.
  - El atribut *configChanges* ha de especificar que no es pugui mostrar el teclat en pantalla al canviar la orientació del mòbil.
- Element <intent-filter>: En aquest punt es on s'especifica quins intents estan enllaçats a l'aplicació. En aquest cas especifiquem dos tipus:
  - El element <*action*> diu que aquesta activitat es la entrada principal.
  - El element <*category*> especifica que l'activitat ha de ser afegida al llançador d'activitats.

4.Desenvolupament de l'aplicació

4.1 Anàlisis

El principal objectiu del projecte es la creació d'un reproductor musical capaç de cercar diferent informació per la xarxa mentre escoltes la teva música emmagatzemada a la targeta SD.

Amb aquesta finalitat, en aquest pròxim apartat s'exposen una llista dels requisits necessaris per que l'aplicació pugui complir les diferents expectatives i a continuació els diferents casos d'us per a un usuari.

4.1.1 Anàlisi de requisits

Requisits funcionals

A continuació es descriuen els diferents requisits de l'aplicació:

- L'aplicació ha de tenir diferents menús intuïtius per donar facilitats al usuari.
- Ha de ser capaç d'escanejar la targeta SD per trobar totes les cançons.
- Ha de ser capaç d'escanejar la targeta SD per mostrar els artistes de les diferents cançons.
- Ha de ser capaç de reproduir aquestes cançons.
- Tenir totes les característiques pròpies d'un reproductor *play*/*pause*/*next*/*previous*, *shuffle*, *repeat*.
- Parar la música en cas de rebre trucades al telèfon.
- Posar l'aplicació en segon pla i que sigui capaç de seguir reproduint música.
- Buscar per Internet la informació bàsica del artista biografia, etc.

- Buscar per Internet els top àlbums del artista així com les cançons d'aquests.
- Buscar per Internet les 5 cançons top del artista.
- Buscar els 5 artistes més semblants al actual.
- Mostrar per pantalla quines d'aquestes cançons buscades en els dos apartats anteriors es troben en la targeta SD i quines no.
- Buscar un vídeo al YouTube de les cançons que no es troben a la SD.
- Buscar per Internet els pròxims esdeveniments en viu del artista i localitzar-los en un mapa.
- Guardar en una base de dades tota la informació anterior per no tenir que buscar constantment a Internet.
- Calcular quin es l'esdeveniment més pròxim a la teva localització.
- Mostrar tota la informació dels esdeveniments.
- Afegir l'esdeveniment al calendari amb una alerta.
- Interaccionar amb Twitter per mostrar que estàs interessat amb l'esdeveniment.
- Mostrar diferents missatges d'error.

Requisits no funcionals

- **Rendiment:** Ha de tenir una execució fluida, encara que trigui una mica en descarregar algunes dades d'Internet. Una vegada descarregades accedirà directament a la base de dades per augmentar aquest rendiment.
- **Usabilitat:** Ha de ser intuïtiu d'utilitzar i tenir menús de fàcil accessibilitat.
- **Compatibilitat:** Intentar que sigui compatible amb el major numero de versions possibles.

4.1.2 Casos d'us

En aquest apartat s'explicaran els principals casos d'us per tal de entendre les possibilitats que ofereix l'aplicació.

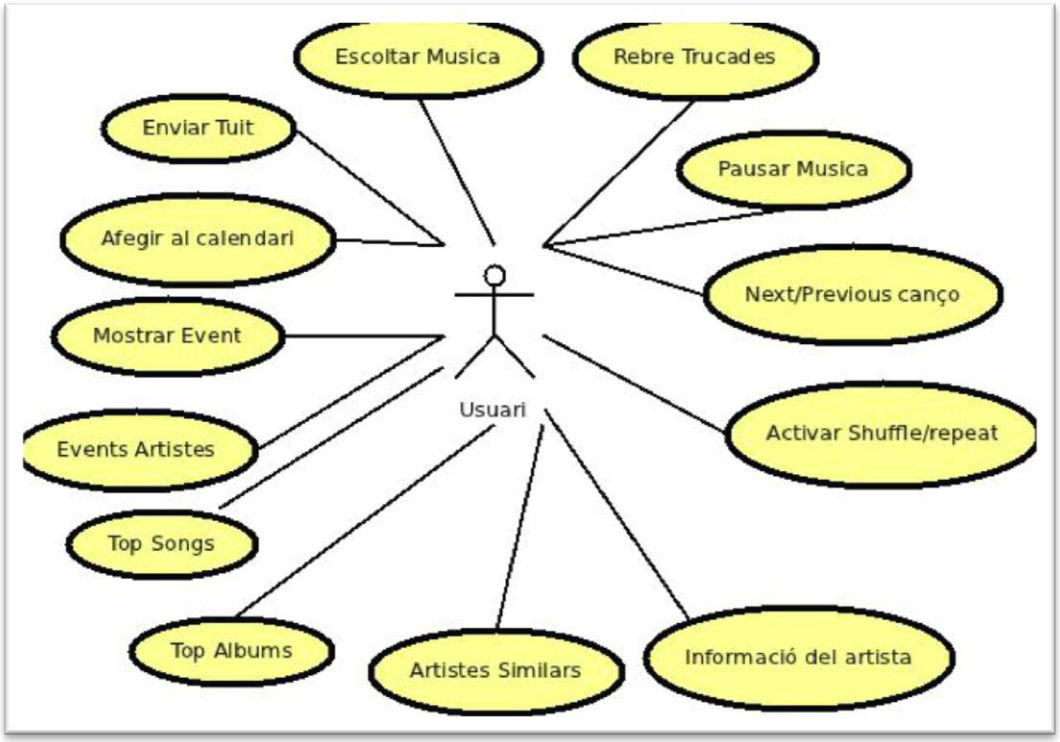


Figura 13: Diagrama Casos d'us

Cas 1: Escoltar Musica

Descripció: El usuari selecciona una cançó i comença la reproducció.

Actor Principal: Usuari

Flux Bàsic

1. El usuari selecciona una cançó a la llista de cançons.
2. L'aplicació obre el reproductor i comença la reproducció de música.
3. L'aplicació crea una notificació a la barra superior del mòbil.
4. L'aplicació mostra la informació del artista així com el nom de la cançó.

Flux Alternatiu

1. El usuari pica el botó back del mòbil. Es torna a la llista principal, però la música segueix reproduint-se
2. El usuari pica al botó home del mòbil. L'aplicació torna al escriptori de Android, però la música segueix reproduint-se.

Cas 2: Rebre Trucades

Descripció: El usuari està escoltant música i rep una trucada.

Actor Principal: Usuari

Flux Bàsic

1. El usuari selecciona una cançó de la llista.
2. L'aplicació obre el reproductor, activa el servei i comença la reproducció de música.
3. L'aplicació crea una notificació a la barra superior del mòbil.
4. L'aplicació mostra la informació del artista així com el nom de la cançó.
5. El dispositiu rep una trucada i la música es pausa.
6. L'usuari finalitza i continua la reproducció de música

Flux Alternatiu

1. El usuari pica el botó back del mòbil. Es torna a la llista principal, però la música segueix reproduint-se
2. El usuari pica al botó home del mòbil. L'aplicació torna al escriptori de Android, però la música segueix reproduint-se.

Cas 3: Pausar Música

Descripció: El usuari està escoltant música i pausa la música.

Actor Principal: Usuari

Flux Bàsic

1. El usuari selecciona una cançó de la llista.
2. L'aplicació obre el reproductor, activa el servei i comença la reproducció de música.
3. L'aplicació crea una notificació a la barra superior del mòbil.
4. L'aplicació mostra la informació del artista així com el nom de la cançó.
5. El usuari pica sobre el botó de pause.
6. L'aplicació pausa la música i canvia el botó pel del play.

Flux Alternatiu

1. El usuari pica el botó back del mòbil. Es torna a la llista principal, però la música segueix reproduint-se
2. El usuari pica al botó home del mòbil. L'aplicació torna al escriptori de Android, però la música segueix reproduint-se.
3. El usuari rep una trucada i es pausa la música.

Cas 4: Next/Previous Song

Descripció: El usuari està escoltant música i passa a la següent o anterior cançó.

Actor Principal: Usuari

Flux Bàsic

1. El usuari comença la reproducció d'una cançó.
2. El usuari pica el botó *next* i comença la següent cançó.
3. El usuari pica el botó *previous* i comença l'anterior cançó.

Flux Alternatiu

1. El usuari pica el botó back del mòbil. Es torna a la llista principal, però la música segueix reproduint-se
2. El usuari pica al botó home del mòbil. L'aplicació torna al escriptori de Android, però la música segueix reproduint-se.
3. Acaba la reproducció sencera de la cançó i automàticament es passa a la següent.

Cas 5: Activar Shuffle/Repeat

Descripció: El usuari està escoltant música i activa la opció Shuffle/Repeat.

Actor Principal: Usuari

Flux Bàsic

1. El usuari comença la reproducció d'una cançó.
2. El usuari activa l'opció shuffle.
3. La següent cançó serà una aleatòria i no la següent de la llista.
4. El usuari desactiva l'opció shuffle.
5. El usuari activa l'opció repeat.
6. Es repetirà la cançó actual fins que es desactivi l'opció.

Flux Alternatiu

1. El usuari pica el botó back del mòbil. Es torna a la llista principal, però la música segueix reproduint-se
2. El usuari pica al botó home del mòbil. L'aplicació torna al escriptori de Android, però la música segueix reproduint-se.

Cas 6: Informació del artista

Descripció: El usuari vol llegir informació sobre un artista.

Actor Principal: Usuari

Flux Bàsic

1. El usuari a través de la pestanya d'artistes obre la llista d'artistes disponibles.
2. El usuari selecciona un artista de la llista.
3. L'aplicació descarrega de Last.fm la informació relacionada amb l'artista.(un cop)
4. L'aplicació guarda la informació en una base de dades.
5. Es mostra per pantalla la biografia del artista.

Flux Alternatiu: Des de el reproductor de música

1. El usuari està a la pantalla del reproductor escoltant una cançó.
2. Pica sobre el nom del artista que apareix a la pantalla.
3. L'aplicació descarrega de Last.fm la informació relacionada amb l'artista.(un cop)
4. L'aplicació guarda la informació en una base de dades.
5. Es mostra per pantalla la biografia del artista.

Flux Alternatiu

1. El pas 3 no es farà si tenim la informació guardada a la base de dades.
2. Si no hi hagués connexió a Internet l'aplicació tornaria al menú principal i no es podria accedir a la secció.

Cas 7: Artistes Similars

Descripció: El usuari vol mirar quins artistes son similars.

Actor Principal: Usuari

Flux Bàsic

1. El usuari realitza tots els passos del Cas 6.
2. A la pantalla de la biografia el usuari pica al botó menú.
3. L'aplicació mostra l'opció de veure els artistes semblants.
4. El usuari pica i l'aplicació mostra en una llista els 5 artistes semblants a ell.
5. Si pica sobre un dels artistes es farà el procediment del Cas 6.

Cas 8: Top Àlbums

Descripció: El usuari vol veure els top àlbums d'un artista.

Actor Principal: Usuari

Flux Bàsic

1. El usuari realitza tots els passos del Cas 6.
2. El usuari obre la pestanya de top àlbums al perfil del artista
3. L'aplicació descarrega els àlbums de Last.fm. (Un cop)
4. L'aplicació guarda a la base de dades la informació descarregada.
5. Es mostra la llista de àlbums en pantalla.
6. L'usuari sobre un element de la llista.
7. Es mostra la informació del àlbum i les seves cançons.
8. Si les cançons es troben a la targeta SD es mostraran en verd, sinó tindrem un enllaç al vídeo a YouTube.
9. Si es selecciona una cançó en verd podrem escoltar les cançons del àlbum que tinguem a la targeta SD.

Flux Alternatiu

1. Si no hi hagués connexió a Internet no podríem accedir a la secció.

Cas 9: Top Songs

Descripció: El usuari vol veure les 5 millor cançons del artista.

Actor Principal: Usuari

Flux Bàsic

1. El usuari realitza tots els passos del cas 6.
2. El usuari obre la pestanya de Top songs al perfil del artista.
3. L'aplicació descarrega les 5 millors cançons de Last.fm.(1 cop).
4. L'aplicació guarda les cançons a la base de dades.
5. Es mostra les 5 millors cançons en una llista a la pantalla.
6. Es mostraran en verd si el usuari les tingués a la targeta SD, sinó es mostrara un enllaç al YouTube..
7. Si es selecciona una cançó en verd es podran escoltar totes les cançons que es tinguin del artista.

Flux Alternatiu

1. Si no hi hagués connexió a Internet no podríem accedir a la secció.

Cas 10: Esdeveniments del artista

Descripció: El usuari vol veure els pròxims esdeveniments del artista.

Actor Principal: Usuari

Flux Bàsic

1. El usuari realitza tots els passos del cas 6.
2. El usuari obre la pestanya de esdeveniments al perfil del artista.
3. Es descarreguen els esdeveniments de Last.fm(1 cop).
4. Es guarden els esdeveniments a la base de dades.
5. L'aplicació mostra en un mapa tots els esdeveniments així com la teva posició actual.
6. Es mostra el esdeveniment més a proper.

Flux Alternatiu

1. Si no hi hagués connexió a Internet no podríem accedir a la secció.

Cas 11: Mostrar Informació esdeveniment

Descripció: El usuari vol veure informació detallada d'un esdeveniment.

Actor Principal: Usuari

Flux Bàsic

1. El usuari realitza tots els passos del cas 6.
2. El usuari realitza tots els passos del cas 10.
3. El usuari pica sobre un dels esdeveniments situats sobre el mapa o el esdeveniment més proper.
4. L'aplicació mostra per pantalla tota la informació detalla del esdeveniment.

Flux Alternatiu

1. Si no hi hagués connexió a Internet no podríem accedir a la secció.

Cas 12: Afegir al calendari

Descripció: El usuari vol veure els pròxims esdeveniments del artista.

Actor Principal: Usuari

Flux Bàsic

1. El usuari realitza tots els passos del cas 6.
2. El usuari realitza tots els passos del cas 10.
3. El usuari realitza tos els passos del cas 11.
4. El usuari pica al botó mostrat de afegir al calendari.
5. L'aplicació mostrarà un menú de selecció de calendari.
6. L'aplicació afegirà un recordatori al calendari que es mostrarà el dia abans.

Flux Alternatiu

1. Si no hi hagués connexió a Internet no podríem accedir a la secció.

Cas 13: Enviar un Tuït

Descripció: El usuari vol enviar un tuït amb la informació relacionada del concert.

Actor Principal: Usuari

Flux Bàsic

1. El usuari realitza tots els passos del cas 6.
2. El usuari realitza tots els passos del cas 10.
3. El usuari realitza tos els passos del cas 11.
4. El usuari picarà al botó de connexió amb xarxes socials.
5. El usuari escollirà l'opció de Twitter.
6. L'aplicació obrirà un navegador per tal de que el usuari es pugui autenticar a Twitter.
7. El usuari s'autenticarà a Twitter i s'enviarà el tuït al seu compte.
8. L'aplicació tornarà a la pantalla del esdeveniment.

Flux Alternatiu

1. Si no hi hagués connexió a Internet no podríem accedir a la secció.
2. Si no es tingués compte a Twitter no es podria enviar.
3. No es podrà enviar el mateix tuït dos cops seguits

4.2 Disseny de l'aplicació

En aquest apartat s'explicarà detalladament com s'ha desenvolupat l'aplicació. Primer de tot es mostrarà un diagrama de classes simplificat per poder entendre l'estructura general de l'aplicació.

A continuació es mostrarà l'estructura de la base de dades dissenyada per guardar les dades mitjançant el corresponent diagrama d'entitat/relació i a continuació s'explicarà la metodologia de desenvolupament.

4.2.1 Diagrama classes general

A la figura 3.7 podrem ver el diagrama classes general de l'aplicació. Es una versió simplificada per poder entendre la estructura bàsica del funcionament de l'aplicació. Més endavant s'explicarà amb una mica de detall algunes parts importants de l'aplicació.

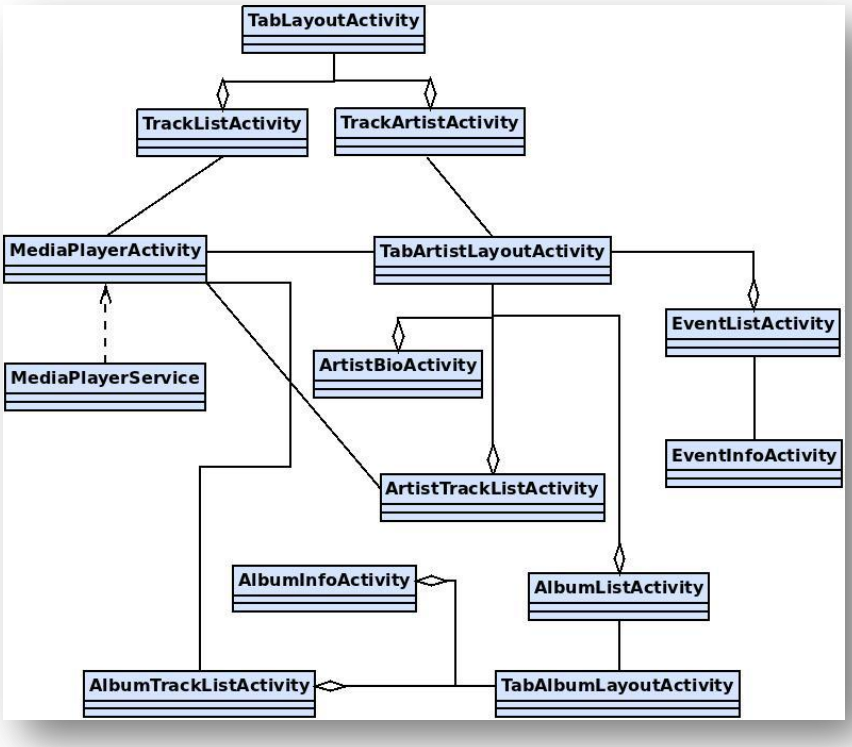


Figura 14: Diagrama de Classes general

4.2.2 Diagrama E/R de la base de dades

L'aplicació també compta amb una petita base de dades creada SQLite que es la eina utilitza Android per a la creació d'elles. La base de dades servirà per guardar la informació que anem descarregant d'Internet i així no tenir que anar a buscar la informació a Internet tota l'estona. A la figura 3.8 podem veure el diagrama E/R que ens mostra la relació entre les diferents taules.

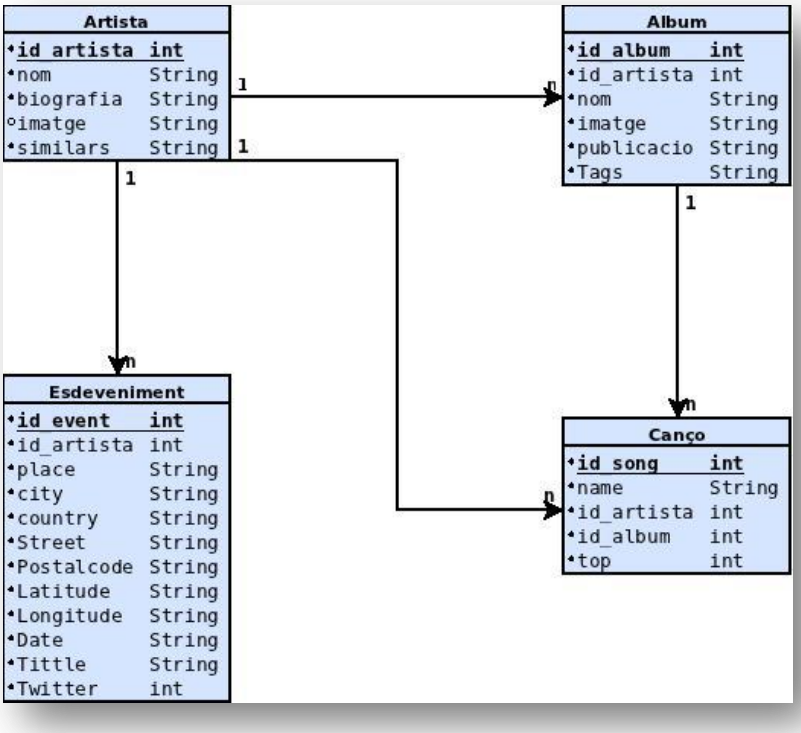


Figura 15: Diagrama E/R de la base de dades

Com es pot veure en la figura la base de dades es molt senzilla i està composada per quatre taules simples. Servirà per guardar tot lo relacionat amb la informació dels artistes.

4.3 Metodologia de Desenvolupament de l'aplicació

En aquest apartat s'explicarà la metodologia a seguir en el desenvolupament de l'aplicació per Android i es mostrarà el desenvolupament d'algunes de les parts més importants de l'aplicació, mostrant a la vegada una sèrie de captures de pantalla fetes directament des de el dispositiu on s'han realitzat les diferents proves.

4.3.1 Pantalla principal. *TabLayoutActivity*.

El primer pas de desenvolupament de l'aplicació va ser la creació de la pantalla principal. En el cas de la nostra aplicació el *layout* utilitzat va ser el anomenat *TabLayout*. Aquest *layout* es una vista amb un tipus de navegació per pestanyes(*Tabs*). En aquesta pantalla disposarem de dues pestanyes la de cançons i la llista d'artistes. Més endavant utilitzarem el mateix *layout* per la vista de la part d'informació dels artistes.

Aquest *Layout* està format per diferents elements**.** **(ANNEX 7.1, CODI 7.1)**

- ***TabHost*:** Es el element pare del *Layout*. Dins d'ell estarà la definició dels seus fills que en aquest cas seran les diferents pestanyes.
- ***TabWidget*:** Es el element fill principal del *TabHost*. Aquest element serà principalment les pestanyes.
- ***FrameLayout*:** Aquest element principalment serà el contingut de cada pestanya. Es on es carregaren les activitats.

Un cop hem desenvolupat la part en XML, s'ha de desenvolupar una senzilla part en codi Java per tal de poder afegir les accions que realitzarà cada pestanya.

Primer de tot hem recuperar el objecte pare que es el *TabHost* aquesta acció la realitzem amb el mètode que heretem de la classe *TabActivity* `getTabHost()`. Una vegada ho recuperem hem de crear la nova pestanya amb el mètode `newTabSpec()`, li inserim la informació bàsica com el icona que volem mostrar així com el nom. Finalment creem un Intent per tal de carregar l'activitat que vulguem dins del *frame* que hem creat al XML. Una vegada definit tot inserim la pestanya dins del *TabHost* pare amb el mètode `addTab()`. **(ANNEX 7.1, CODI 7.2)**

4.3.2 Pestanya Principal. Llista de cançons.

Una vegada explicada la pantalla principal que apareix al iniciar l'aplicació, s'explicarà la pestanya activa per defecte. Aquesta pestanya ens mostra la llista de les cançons disponibles a la nostra targeta SD.

Aquesta activitat utilitza un tipus de layout en forma llista de ítems. Aquesta vista es anomenada *ListView* i ens mostra una llista en una columna movible. Aquesta vista està desenvolupada una petita part d'ella en XML i majoritàriament en Java.

El seu desenvolupament en format XML destaca per la seva senzillesa, ja que només hem de crear un tipus de objecte anomenat ListView amb els atribut corresponents. Una vegada creat el XML aquesta llista serà modificada en el Java per tal d'afegir els elements corresponents.

Abans de entrar més en detall de com omplir aquesta llista amb els elements corresponents, primer de tot ens fixarem de com carreguem les cançons de la targeta SD. Aquestes cançons seran els elements que mostrarem a la llista.

Per poder extreure la informació de la targeta SD utilitzem una classe especial que hereta de *AsyncTask.* Aquesta classe es divideix en quatre passos fonamentals i es la que utilitzarem en la majoria d'activitats de l'aplicació per tal de carregar totes les dades i llistes que apareixen. AsyncTask es una tipus de classe que ens permet realitzar diferents tasques en segon pla. Aquests quatre passos son els següents:

1. ***onPreExecute():*** Aquesta tasca es cridada pel *Thread* de la UI abans que la tasca principal s'executi. Serveix per poder mostrar una barra de progres a la interfície.
2. ***doInBackGround():*** Funció cridada en *background* una immediatament després de la primera funció. Es la feina que es farà en el rerefons de l'aplicació.
3. ***onProgressUpdate():*** Utilitzada per mostrar qualsevol formulari a la pantalla de l'aplicació mentre la informació s'està processant en el rerefons.
4. ***OnPostExecute():*** Mètode cridat una vegada que el pas ha finalitzat el seu procés. El resultat del pas 2 es passat cap aquest punt per mostrar els resultats.

Aquest mètode l'utilitzarem per carregar qualsevol informació de l'aplicació ja sigui treta de la targeta SD ,d'Internet o de la pròpia base de dades. En la nostra aplicació utilitzarem el primer cas per mostrar una barra de progres, el segon cas per agafar la informació i el quart cas per mostrar els resultats per pantalla.

Una vegada explicat aquest procés, seguim amb la càrrega de cançons i de com omplir aquest tipus de vista especial.

En un primer moment es va decidir per poder carregar les cançons de la targeta SD anar explorant carpeta per carpeta i agafar els fitxers que tinguessin extensió .mp3, es va comprovar que aquest procés era lent i necessitava de recursivitat per tal de funcionar,

per tant es va decidir buscar una altra forma més eficient. Investigant per Internet diferents pàgines sobre el tema es va trobar una interfície *d'Android* que ja et feia aquest procés.

Aquesta interfície s'anomena *MediaStore* i es un proveïdor de continguts que et retorna la informació seleccionada mitjançant diferents *URIs*. Per exemple en el nostre cas es va tenir que cridar les següents *URIs*.

- **`MediaStore.Audio.Media.TITLE`**
- **`MediaStore.Audio.Media.ARTIST`**

Aquesta manera d'extreure la informació sobre les cançons de la targeta SD era molt més eficient i molt més senzilla de desenvolupar. Per poder extreure la informació s'ha de cridar un mètode intern d'Android anomenat managedQuery(). Aquest mètode es senzill ja que s'utilitza amb uns paràmetres semblants com si es tractés de codi SQL, en aquest cas seria com un SELECT. Els quatre paràmetres que li passarem seran:

(ANNEX 7.2, CODI 7.3)

- **Taula:** `MediaStore.Audio.Media.EXTERNAL_CONTENT_URI.` Ens indica que hem de buscar la informació a la targeta SD externa.
- **Columnes:** Les mencionades anteriorment. Nom i artista del fitxer.
- **Where:** `Audio.Media.DATA + " like ? OR " + Audio.Media.DATA + " like ? ",new String[]{"%mp3"}.` Aquesta sentencia significa que volem que només escanegi aquells fitxers amb extensió .mp3.
- **Order By:** En el nostre cas ordenarem pel títol del fitxer.

Una vegada tenim tota la informació extreta de la targeta SD hem de mostrar el títol i el autor a la llista del *Layout*.

Per fer aquest procediment s'utilitza un tipus de *Array* especial anomenat *ArrayAdapter*. Per defecte aquest *ArrayAdapter* crea una vista per cada *item* de la llista, els elements extrets de la targeta SD es posen en una llista, cridant al mètode *toString()* posant aquest *String* retornat dins d'un element *TextView*. En el nostre cas es va tenir que crear un *ArrayAdapter* personalitzat per tal de que es mostrés el nom de la cançó i una mica més petit el nom del artista.

Aquesta nova classe creada va ser anomenada *SongArrayAdapter* i hereta de la classe *ArrayAdapter<String>*. Per poder crear cada ítem de la llista s'han de crear dos *TextView* per tal de mostrar aquestes dades. Una vegada creat aquest nou ArrayAdapter hem de mostrar el seu contingut a la vista.

Figura 16: Llista de cançons

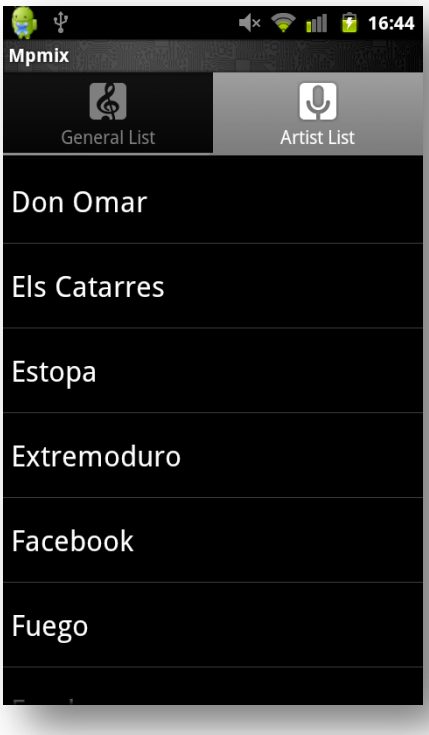


Figura 17: Llista d'artistes

4.3.3 Reproductor musical.

En aquest apartat s'explicarà el funcionament bàsic del reproductor de música. Una vegada seleccionem una cançó de la llista explicada anteriorment s'obrirà el reproductor musical.

Aquesta part està formada per dues classes principals una activitat que serà la que crearà la interfície gràfica e interactuarà amb els controls de la pantalla i una segona classe que serà un servei, aquesta segona ens permetrà seguir escoltant música si es minimitza l'aplicació amb el botó home.

Activitat. MediaPlayerActivity

El primer pas de tots serà crear la interfície gràfica, es desenvoluparà igual que en apartats anteriors, per poder tenir els controls bàsics d'un reproductor musical: *Play, Pause, Next, Previous, Shuffle, Repeat*, etc. Una vegada creats e iniciats els hi afegim els *listeners* corresponents per tal de que realitzin les accions corresponents.

Una vegada creada tota la interfície hem de connectar aquesta activitat amb el nostre servei, explicat en el següent apartat, que serà l'encarregat de carregar les cançons i realitzar les accions del *MediaPlayer.* Aquesta acció la realitza el mètode de l'activitat anomenat doBindService() **(ANNEX 7.3, CODI 7.4).**

Amb aquest mètode realitzarem la connexió sobre el servei amb els paràmetres següents:

- `serviceIntent`: Intent que es crearà al principi de la activitat i que ens permetrà comunicar l'activitat i el servei.
- `serviceConnection`: Instància de la classe *serviceConnection* que ens indicarà que accions haurà de fer el servei quan es connecta o desconnecta.

- `Context.BIND_AUTO_CREATE:` crear automàticament el servei, sempre que la unió existeix

Figura 18: Reproductor musical

Servei. MediaPlayerService

Com s'ha explicat en un apartat anterior el servei es podria tractar com si fos un procés. Aquests seguiran executant-se encara que no hi hagi cap interfície gràfica.

Una vegada que creem un servei es necessari sobreescriure uns mètodes per tal de que el servei realitzi les accions que nosaltres desitgem. Els mètodes a sobreescriure son els següents:

- onStartCommand(): Es el primer mètode que es crida quan una activitat es connecta al servei cridant al mètode *startService()*. Una vegada aquest mètode es cridat el servei es iniciat i pot executar-se indefinidament en segon pla. En el mètode que es va sobreescriure es realitzen les següents accions al iniciar el servei.

  En el nostre cas les accions que realitzarem dins d'aquest mètode seran les següents. Primer de tot s'iniciarà el control del telèfon, on activarem un *listener* per poder realitzar accions quan rebem una trucada.

  Primer de tot hem d'activar el servei que començarà a escoltar l'estat del telèfon. El *listener* mirarà dos casos:

  Si el telèfon rep una trucada(`TelephonyManager.CALL_STATE_RINGING`) el reproductor es posarà en pausa i es podrà escoltar la trucada amb facilitat i el segon cas es quan s'acabi la trucada (`TelephonyManager.CALL_STATE_IDLE`) el reproductor seguirà el seu funcionament normal.

  El segon pas que es realitzarà serà la creació d'un *Broadcast* que enviarà missatges a l'activitat per tal de que aquesta actualitzi la barra de progres de la cançó, ja que l'activitat es l'encarregada de la UI, per tal de que avanci mentre sona la cançó. Aquest *broadcast* també s'encarregarà de canviar el nom de la cançó de la pantalla quan aquesta canviï. **(ANNEX 7.3, CODI 7.5).**

  Amb aquest sistema de creació *Broadcast* podrem enviar missatges entre el servei i l'activitat, i al inversa.

Una vegada tenim tots aquests *listener* actius tindrem que carregar el *media Player* i les cançons per que aquestes puguin ser reproduïdes. El *media Player* d'Android es gestiona com una màquina d'estats. El següent diagrama ens mostra el seu cicle de vida.

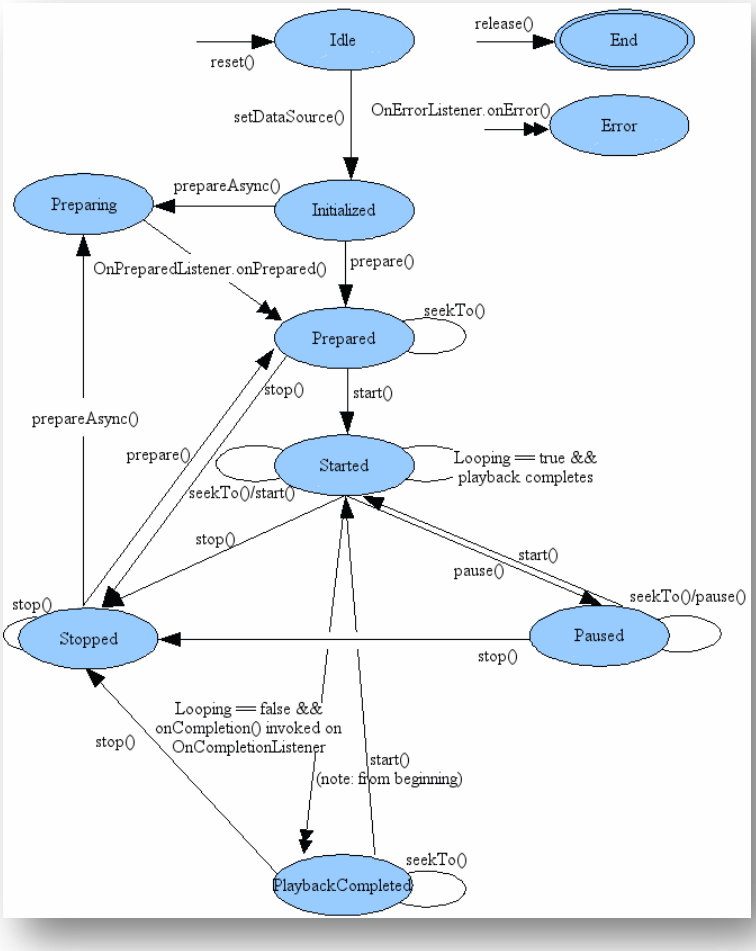


Figura 19: Diagrama d'estat Media player

Els ovals del diagrama representen els estats on el reproductor pot estar en un cert moment de la seva vida. Els arcs representen les operacions de control de reproducció que impulsen la transició d'un estat. Son els canvis entre estats. Hi ha dos tipus de arcs. Els arcs amb una fletxa amb un cap representen crides a mètodes síncrons, mentre els estats amb fletxa de dos caps representen crides a

mètodes asíncrons. El nostre servei seguirà el diagrama d'estats anterior per tal de que funcioni correctament.

- ***onBind():*** El sistema crida aquest mètode quan un component vol connectar-se amb el servei cridant al mètode bindService(). Sempre s'ha de implementar aquest mètode sinó es retornara null.
- ***onCreate():*** El sistema crida aquest mètode quan es creat per primera vegada. Si el servei ja s'està executant aquest mètode no es cridarà.
- ***OnDestroy():*** El sistema crida aquest mètode quan el servei ja no s'està utilitzant i ha de ser destruït. Aquest mètode ha de desactivar tots el Threads obert i tancar els listeners.

Una vegada explicat el procés general del servei de reproducció de música s'explicarà una altra funció important que realitza. Aquesta funció es la creació de la notificació a la barra superior del dispositiu.

Aquesta notificació s'activa quan es comença la reproducció de la música i es desactiva cada vegada que es pausa o es para definitivament la música.

Figura 20: Notificació barra d'estat

Per poder crear aquesta notificació primer hem de recuperar el *manager* de notificacions de les propietats del sistema. Una vegada recuperat creem un intent amb l'activitat que s'executarà quan piquem sobre ella, en el nostre serà la *MediaPlayerActivity*. Per últim definim la icona que es mostrarà i el text que volem mostrar a sota que serà el nom de la cançó actual. **(ANNEX 7.3, CODI 7.6).**

4.3.4 Cerca d'informació a Internet.

En aquest apartat s'explicarà com l'aplicació cerca la informació dels artistes per Internet, en quin format es retornat i com es llegeix a l'aplicació per tal de que surti en un format correcte a l'aplicació.

Primer de tot es va tenir que buscar per Internet una API que no fos de pagament per poder descarregar la informació més completa dels artistes. Es van estar mirant diverses pàgines que ho oferien però les més completa i simple va ser la de *Last.fm*.

Last.fm es una pàgina que ofereix molta informació sobre el món de la música i la seva API permet a qualsevol usuari que estigui desenvolupant una aplicació utilitzar les seves dades. La seva Api també permet pujar contingut al perfil d'usuari personal de *Last.fm* però aquesta part no la necessitarem.

El primer pas de tots abans de posar-se a cridar els seus mètodes de cerca d'informació va ser el registre a la pàgina i aconseguir una api *key* que ens permetés accedir als seus mètodes.

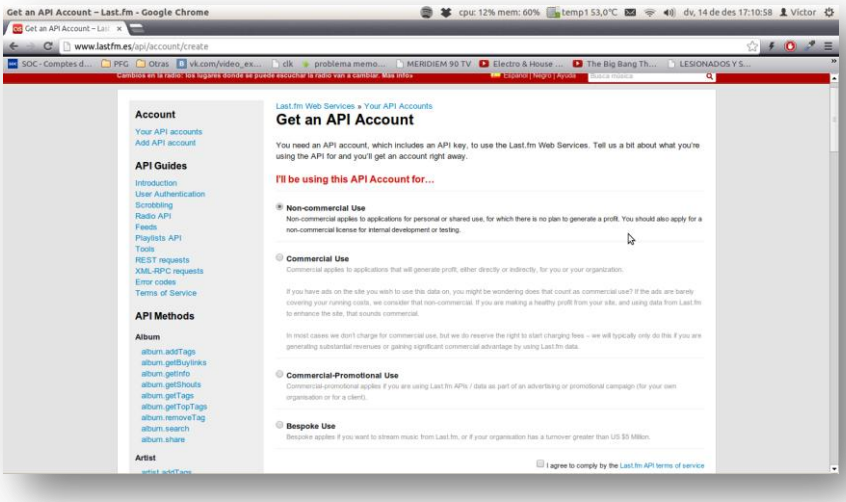


Figura 21: Registre a Last.fm

Una vegada realitzat el correcte registre de l'aplicació a la pàgina aquesta ens proveirà amb una clau per poder utilitzar tots els mètodes de la seva API sense obligar al usuari que utilitza l'aplicació a registrar-se. Només estaria obligat a registrar-se si la nostra aplicació utilitzes els mètodes d'actualització del perfil personal (Scrobling).

Per poder cridar a un mètode en concret de la api funciona per URL, un exemple seria la següent:

http://ws.audioscrobbler.com/2.0/?method=artist.getinfo&artist=Cher&api_key=74b187db4bb3fb0238c23dd85729af70

Aquesta URL està formada per diferents parts, la primera es el *method* on se li posa la funció a utilitzar en aquest cas concret serà *artist.getinfo,* en un segon terme i separat sempre per '&' vindria el artista, ja que la informació que estem buscant està relacionada amb l'artista, en aquest cas es *Cher* i per últim seria el numero de api que hem aconseguit. Per cada funció hi ha una petita explicació a la pàgina Web de com utilitzar-les.

Una vegada feta crida la pàgina ens retornarà una XML que tindrem que llegir amb Java.

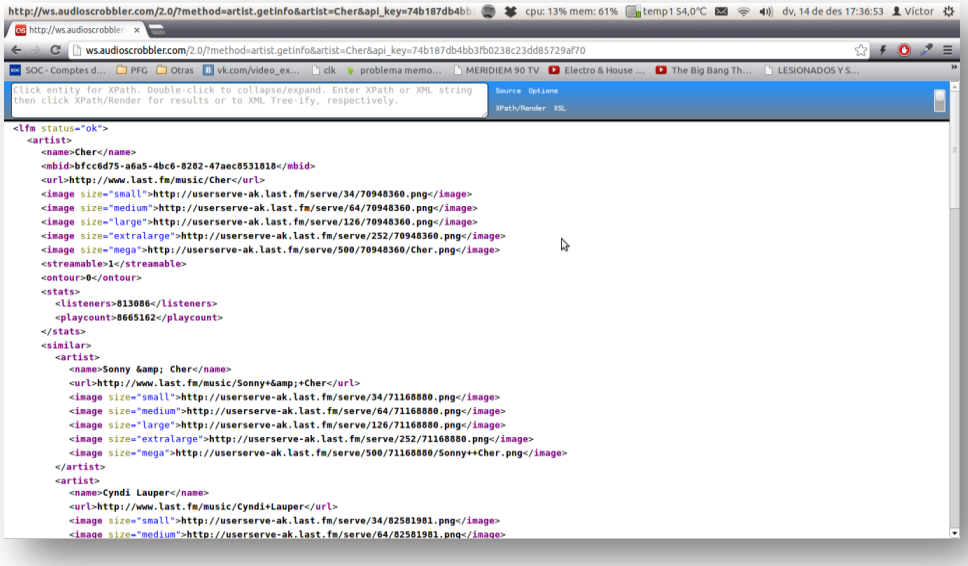


Figura 22: Resultat de crida a Last.fm

S'ha de dir que cada funció de la API de *last.fm* donarà un *xml* semblant però a la vegada diferent. Això vol dir que l'estructura general serà casi igual però el nom de les etiquetes així com el seu contingut serà totalment diferent.

Una vegada explicat el funcionament de la Api de *Last.fm* toca aplicar aquests conceptes a la programació en Java. Per poder realitzar la cerca d'informació des de Java hem de modificar una mica les dades que volem buscar. Això vol dir que tindrem substituir els espais entre paraules pel seu codi Ascii(%20) així com treure tots els accent que porti.

Una vegada modificat, hem de realitzar una connexió a una URL semblant a l'anterior, cada mètode tindrà la seva URL, amb les dades que es poden treure de l'aplicació. Quan la API de Last.fm ens retorni el XML el llegirem i anirem agafant tots aquells elements que ens interessi mostrar. Les imatges següents mostren alguns resultats de realitzar aquesta cerca.

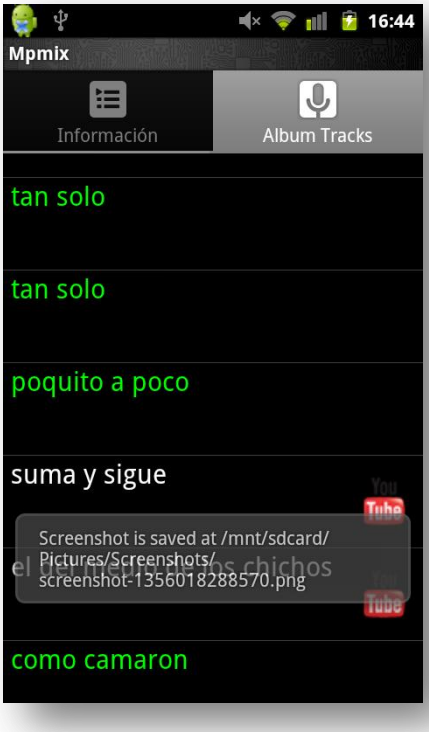


Figura 23: Resultat de cerca d'informació

En la imatge anterior on es mostren diferents cançons es pot observar que tenim algunes marcades en verd i altres no. Aquest fet significa que si la cançó està marcada en verd podrem accedir a ella i poder reproduir-la amb el nostre reproductor ja que la tindrem guardada a la targeta SD, mentre que si no la tenim a la SD ens donarà l'opció de poder veure un vídeo al YoTube de la cançó.

Hem de tenir en compte que per poder realitzar totes aquestes tasques el dispositiu ha de tenir activada la connexió a Internet, 3G o WIFI, si no es tingues la connexió activada saltaria un missatge d'error i no ens deixaria accedir a la secció que requereix d'Internet.

Figura 24: Connexió no disponible

4.3.5 Localització d'esdeveniments musicals.

Una vegada explicat com i d'on cerquem la informació, ens centrem en la localització dels esdeveniments musicals dels artistes. La informació sobre aquets la trobarem a la mateixa font que abans a la API de *Last.fm.* Aquesta informació la localitzarem sobre en un mapa de Google Maps amb la nostre posició actual per tal de poder trobar quin serà el esdeveniment més proper.

Per poder localitzar els esdeveniments en el mapa s'utilitza la API de Google Maps que ens permet mostrar un mapa a la pantalla del dispositiu i també ens permet posar diferents punts sobre ell amb la localització exacta. Aquesta localització es troba mitjançant les coordenades de latitud i longitud.

Creació de la API Key

Abans d'entrar en mes detalls sobre la utilització d'aquesta API de Google hem d'aconseguir una key que ens permeti utilitzar-la. Per poder obtenir-la s'han de seguir una sèrie de passos.

El primer pas de tots es calcular la empremta digital en format MD5 que s'utilitzarà per poder signar l'aplicació final. Aquesta empremta s'ha de introduir en el registre a la pàgina de la API de Google Maps per tal que es pugui associar la clau amb l'aplicació. Mentre es desenvolupa una aplicació en Android, aquesta es signada en mode Debug, significa que l'aplicació es signada per les eines de Debug del SDK. Per poder generar l'empremta hem de localitzar el debug keystore. Aquesta localització varia segons la plataforma, en el nostre cas seria: ~/.android/debug.keystore.

Una vegada he recuperat l'empremta digital podem accedir a la web de *Google* i ens donarà accés complet a la API de *Google Maps* per poder depurar les nostres aplicacions.

Figura 25: Pàgina web de Api de Google

Aquesta *Api Key* serà útil només quan la aplicació estigui en procés de desenvolupament ja que quan es pengi l'aplicació al *Google Play* tindrà que ser signada amb una nova clau privada.

Creació del mapa

Una vegada realitzats aquest passos previs la introducció de mapes dins de la nostra interfície gràfica es una tasca bastant senzilla. Per incloure un mapa a la nostra aplicació utilitzarem el control dins del nostre XML anomenat *MapView*, aquest s'afegeix al *layout* igual que tota la resta d'elements, la peculiaritat més important es que tindrem que afegir el atribut *androi:apiKey* amb la clau obtinguda anteriorment.

Una vegada creat el *layout* amb el mapa hem de afegir-lo dins d'una activitat. Aquest *layout* té la particularitat de que només es possible afegir-lo en tipus especial d'activitat, aquesta s'anomena *MapActivity*.

Una vegada creada l'activitat heretarà de la classe MapActivity aquesta ens obliga a implementar un mètode anomenat isRouteDisplayed(), aquest mètode retornarà True només en el cas de representar algun tipus d'informació sobre rutes, en el nostre cas no caldrà implementar-lo. Per finalitzar la creació del mapa cridarem mètode `setBuiltInZoomControls()` sobre la referencia del control del MapView, amb aquesta crida podrem mostra els controls estàndards de zoom sobre el mapa.

Per finalitzar la creació del mapa sobre la interfície gràfica hem de realitzar un últim pas que serà la de donar permisos de connexió a Internet i la d'utilitzar l'API de Google Maps. Aquets permisos es donaran dins del fitxer AndroidManifest.xml

Situar esdeveniments sobre el mapa

Una vegada creat el mapa i situat sobre la interfície el següent pas que realitzarem serà el de situar els esdeveniments sobre el mapa. Per realitzar aquesta tasca crearem una nova classe que heretarà ItemizedOverlay.

El constructor d'aquesta requereix que li passem per paràmetres una imatge, la qual s'utilitzarà per mostra sobre el mapa, de tipus Drawable. Tots aquets ítems seran guardats en un ArrayList**.** **(ANNEX 7.4, CODI 7.7)**

La herència de la classe ens obligarà a crear alguns mètodes que ens serviran per realitzar una sèrie d'opcions. Un d'aquets mètodes es el anomenat onTap() que serà cridat quan l'usuari piqui sobre una de les marques a sobre del mapa. En la nostra aplicació l'acció que es realitzarà al picar sobre una d'aquestes marca serà la d'accedir a una nova activitat on se'ns mostrarà informació més detallada sobre l'esdeveniment i un petit mapa on es mostrarà a peu de carrer el lloc. **(ANNEX 7.4, CODI 7.8)**

Per poder situar tots aquets ítems sobre el mapa necessitarem les dades de latitud i longitud dels punts. Aquestes dades les trobarem a la API de Last.Fm, amb aquestes dues dades tindrem que crear un nou objecte anomenat GeoPoint. Amb aquest objecte nou crearem un ítem, el que hem explicat abans, i el situarem sobre el mapa en el punt exacte del mapa.

Figura 26: Situació dels esdeveniments sobre el mapa

Com es pot observar en la figura anterior els esdeveniments de artista surten situats amb una icona vermella, mentre que la icona verda indica la nostra posició sobre el mapa. S'ha de dir que si el artista no tingues concerts programats en les properes dates se'ns mostraria un missatge d'alerta i no mostraria el mapa.

Figura 27: Alerta de no esdeveniments

Localització de la posició actual

Una vegada tenim tots els esdeveniments situats sobre el mapa toca situar la nostra posició actual i decidir quin dels esdeveniments trobats es el més proper. Android ens dona la possibilitat de accedir als seus mètodes de localització situats al paquet android.location.

Per poder trobar la nostra localització els dispositius Android ens donen dos proveïdors, cada proveïdor amb els seus avantatges e inconvenients, aquets son el GPS i la triangulació de torres de WIFI. Per una part el GPS es una eina molt precisa, però només funciona al exterior i ràpidament consumeix la bateria del nostre dispositiu i no sol trobar la posició tan ràpid com l'usuari voldria. La triangulació de xarxes WIFI determina la nostra posició a través de les torres de xarxes WIFI del nostre voltant i pot funcionar tan en interior com en exterior, actua més ràpid i no malgasta tanta bateria. A partir d'aquestes dades s'haurà d'escollir quina serà l'opció més òptima.

Hi ha casos on l'obtenció de la nostra localització pot arribar a ser una mica complicat, ja que tenim molts factors que ho fan així. Pot haver vegades on la localització sigui errònia o bé poc acurada. Alguns possibles inconvenients son:

- Multitud de fonts d'ubicació.
- Moviment del usuari.
- Variació en la precisió

Una vegada hem analitzat els diferents mètodes per poder obtenir la nostra localització i alguns avantatges com inconvenients passem a mirar com obtenir aquesta posició en la nostra aplicació.

El component principal del paquet es el Location Manager, que ens dona suport a una API per poder determinar la posició i el rumb de la nostra posició, si fos possible. Com es tracta d'un servei d'Android no cal crear cap instancia. S'ha de fer una crida al sistema perquè aquest ens doni una instància d'aquest objecte LocationManager. **(ANNEX 7.4, CODI 7.9)**

Una vegada obtenim aquesta instancia l'aplicació seria capaç de realitzar diverses accions:

- Consultar el Location Provider per obtenir l'última localització.
- Registrar o anul·lar un registre d'actualitzacions periòdiques per trobar la ubicació actual que obtindrem a partir d'un proveïdor.

En el nostre cas per poder obtenir una localització molt més precisa utilitzarem una barreja de les dues opcions.

En Android la obtenció de la localització funciona pel mètode anomenat callback(devolució de trucada). S'ha d'indicar que es vol obtenir actualitzacions de la nostra posició actual mitjançant un listener que començarà el proveïdor que l'aplicació hagi escollit com la millor opció. Android escull la millor opció d'obtenció de dades mitjançant el objecte Criteria que avalua les diferents opcions i quina es la millor. **(ANNEX 7.4, CODI 7.11)**

En aquest procés també podem controlar la freqüència en la que rebem les diferents actualitzacions amb els diferents paràmetres que li passem al locationManager, li podem passar el mínim interval de temps entre actualitzacions i el mínim canvi de distancia entre diferents actualitzacions. Hem de tenir en compte que per poder utilitzar totes aquestes

funcions hem de donar diferents permisos a la nostra aplicació a través del AndroidManifest.xml. En aquest fitxer hem de donar els següents permisos:

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

Aquest seria el procés habitual que tenim per poder trobar la nostra localització, però disposem d'una sèrie d'estratègies que amb les que podem millorar la precisió d'aquesta posició. Aquest nou de model inclou les decisions de quan comencem a escoltar les diferents actualitzacions i quan comencem a utilitzar les diferents dades guardades d'actualitzacions anteriors. El típic flux que s'utilitza es el següent:

1. Executar l'aplicació
2. Començar a escoltar les diferents actualitzacions dels proveïdors
3. Mantenir una millor estimació de les noves actualitzacions.
4. Parar les noves actualitzacions.
5. Agafar la millor estimació de la ubicació anterior.

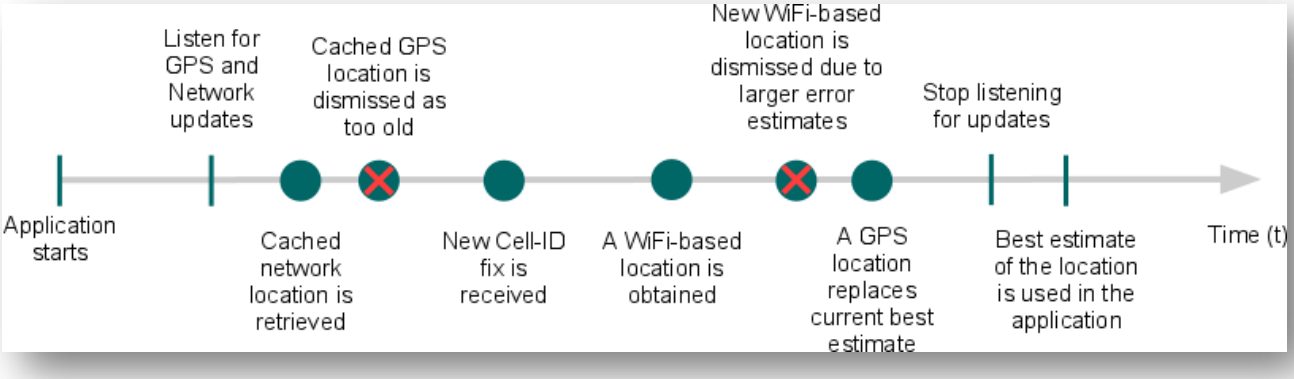


Figura 28: Flux per obtenir localització

Es podria dir que la última que es va trobar sempre serà la més precisa, però com que aquesta localització fixa pot variar en un instant de temps es podria dir que no sempre

serà la mes precisa. Podem incloure una certa lògica en el nostre codi per tal d'escollir la millor localització.

Podem realitzar els següents passos per tal de validar l'exactitud:

- Comprovar si la ubicació recuperada es significativament menor a l'anterior.
- Comprovar si la precisió es millor o pitjor a l'anterior.
- Comprovar quin es el millor proveïdor.

El codi del . **(ANNEX 7.4, CODI 7.12)** mostra un mètode que desenvolupa els passos anteriors.

Localització del esdeveniment mes proper

Una hem aconseguit obtenir la posició més precisa de la nostra localització hem d'intentar aconseguir quin es el esdeveniment més proper a ella. Per poder fer-ho es va implementar la fórmula de Harvesine.

Aquesta fórmula ens permet calcular la distancia entre dos punts dins del cercle màxim del globus terrestre sabent la seva latitud i longitud. Per tant la fórmula tenim es la següent, per qualsevol parell de punts en una esfera:

$$haversin\left(\frac{d}{R}\right) = haversin(\phi 1 - \phi 2) + \cos(\phi 1)\cos(\phi 2)\, haversin\,(\Delta\lambda)$$

On els elements de la fórmula són:

- Haversin es la funció haversine: $haversin(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2}$
- d es la distancia entre els dos punts(distancia esfèrica)
- $\phi 1$ es la latitud en el punt 1
- $\phi 2$ es la latitud en el punt 2

- $(\Delta\lambda)$ es la diferencia de longituds

Hem de tenir en compte que els arguments de la funció haversin han de ser en radians.

Per tant si tenim en compte que:

$$h = haversin\left(\frac{d}{R}\right)$$

$$haversin(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2}$$

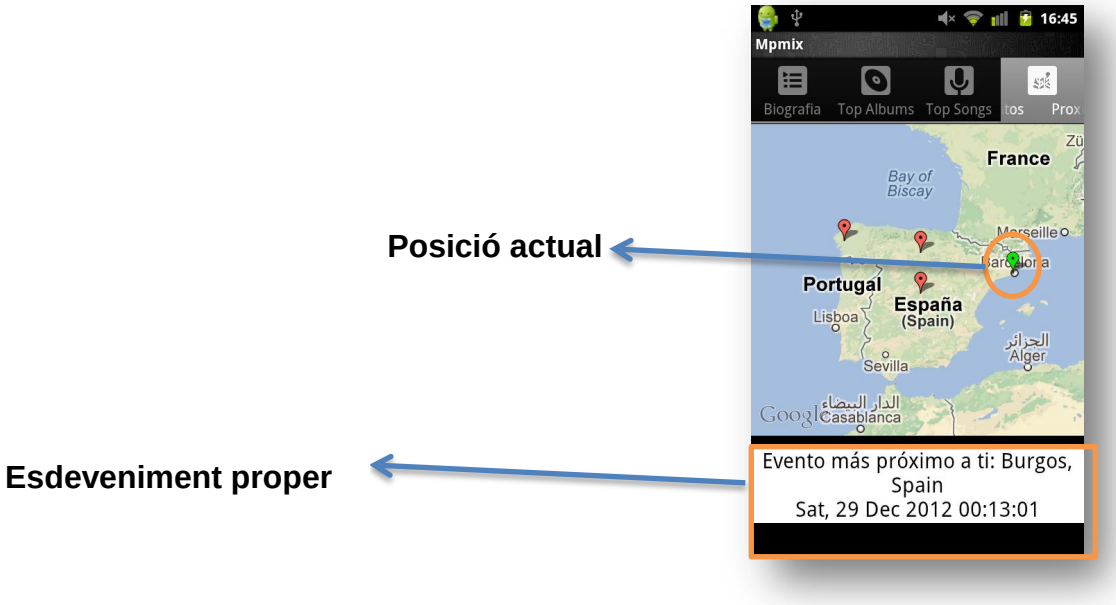
Podrem treure el valor de la distancia entre dos punts sabent que $y = \sqrt{x} \rightarrow y^2 = x$ i que $y = \arcsin(x) \rightarrow x = \sin(y)$ per tant el valor de la distancia serà:

$$h = haversin\left(\frac{d}{R}\right) \rightarrow h = \sin^2\left(\frac{\frac{d}{R}}{2}\right) \rightarrow \sqrt{h} = sin\left(\frac{\frac{d}{R}}{2}\right) \rightarrow \arcsin\left(\sqrt{h}\right) = \frac{\frac{d}{R}}{2} \rightarrow \boldsymbol{d}$$

$$= \boldsymbol{2Rarcsin(\sqrt{h})}$$

On h serà el valor calculat anteriorment.

La implementació es pot trobar a **(ANNEX 7.4, CODI 7.13)**



4.3.6 Pantalla d'esdeveniment musical.

Una vegada tenim tots aquets elements sobre el mapa podem accedir a un esdeveniment en concret per tal de tenir informació més detallada sobre ell i també realitzar una sèrie de tasques.

Figura 29: Pantalla d'esdeveniment.

Alguna de les funcions que podrem fer a partir d'aquesta pàgina son les següents.

Afegir esdeveniment al calendari

Una de les opcions que tindrem en aquesta pantalla serà la de poder afegir el esdeveniment al calendari del dispositiu Android. Per poder accedir a les funcions dels calendaris es fa d'una manera semblant que amb les cançons, a la API de la versió d'Andoid 4.0 es pot accedir a través de diferents mètodes i de manera mes senzilla, a través de URL. En aquest cas la URL a la que accedirem serà la següent

`content://com.android.calendar/events` i per poder crear l'alerta que ens notificarà es `content://com.android.calendar/reminders`.

Per tant a partir d'aquí podrem accedir als calendaris d'Android com si d'una base de dades es tractés. Tindrem la taula *events* per poder inserir el esdeveniment musical i tindrem la taula *reminders* per poder inserir les alertes.

S'ha de tenir en compte que per les versions d'Android no es fa amb aquest mètode sinó que ja existeixen una sèrie de mètodes a la API per tal de manipular els calendaris. Aquesta versió implementada està adaptada per tal de que pogués funcionar a partir de la versió d'Android 2.3.3, mentre que la API de manipulació de calendaris està feta per versions d'Android 4.0. **(ANNEX 7.5, CODI 7.14 I 7.15)**

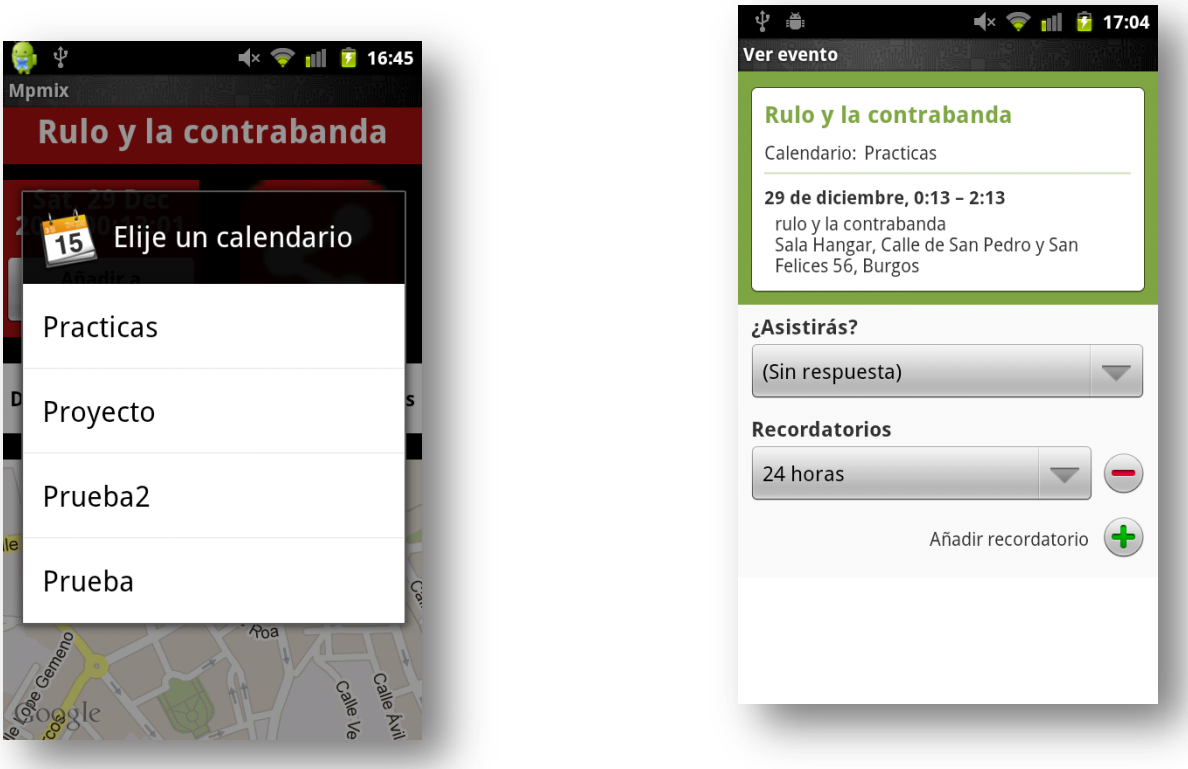


Figura 30: Afegir esdeveniment al calendari

Connexió a xarxes socials

En aquets últims anys les xarxes socials han experimentat un gran desenvolupament i un gran increment en quant a nombre d'usuaris. Twitter i Facebook son dos d'aquestes xarxes socials que han experimentat aquest gran creixement exponencial arribant a milions d'usuaris arreu del món. Per aquest principal motiu es va decidir afegir una petita opció amb la que es pogués connectar amb elles. Per raons de falta de temps només es va poder implementar l'opció de connexió amb Twitter. Aquesta xarxa social ens permet connectar-nos a milions d'usuaris i enviar Tuïts dirigits a ells o compartir-los amb tots els contactes que et segueixin(Followers).

Abans de poder connectar-nos amb Twitter, la xarxa social obliga al desenvolupador a registrar l'aplicació a la web per tal de que es pugui accedir a la API de Login. Una vegada realitzat el registre a la pàgina https://dev.twitter.com/apps/new Twitter et proporciona una clau.

A partir d'aquí ja podem accedir a través de la llibreria de Java twitter4j, amb ella podrem accedir a tots els mètodes de la API de Twitter amb facilitat. El primer que es va tenir que implementar va ser la forma per poder autenticar-nos.

La part d'autenticació a Twitter es la més complexa en quant a implementació, el mètode utilitzat per ella s'anomena OAUTh. Aquest sistema es un protocol obert que ens permet una autorització segura de una API. Es un mètode per poder interactuar amb dades protegides i poder publicar-les. En altres paraules OAuth permet a un usuari d'un lloc A compartir una certa informació a aquest lloc A des de un altra aplicació B sense compartir tota la seva identitat.

En la nostra aplicació l'Activitat que s'encarrega d'aquesta autenticació es l'anomenada **PrepareRequestTokenActivity**, aquesta ens estableix el consumidor i el nostre proveïdor del protocol OAuth i comença una tasca asíncrona que ens permetrà realitzar les accions corresponents. Aquesta tasca ens executarà un nou Intent que serà l'encarregat de obrir una finestra del nostre navegador Web per tal d'efectuar el login correctament. Una vegada ens hem loguejat a Twitter el navegador ens retornarà un altre cop cap a la nostra aplicació mitjançat una URL de callback **x-oauthflow-twitter://callback**. **(ANNEX 7.6, CODI 7.16 I 7.17)**. Una vegada ens hem autenticat al nostre compte hem d'interceptar aquest callback. Una vegada interceptat es pot continuar amb l'execució normal de la nostra aplicación. **(ANNEX 7.6, CODI 7.18).**

A partir d'aquest punt l'aplicació ja podrà enviar el Tuït corresponent i així informar del esdeveniment a la resta dels nostres Followers.

Figura 31: Connexió a Twitter

4.4 Eines utilitzades

En aquest apartat s'explicaran algunes de les eines utilitzades en el desenvolupament del PFG i en concret en el desenvolupament de l'aplicació

4.4.1 Dispositiu Hardware

El dispositiu utilitzat per la realització de les proves es el següent:

HTC Desire

Mida: 119mm x 60mm x 11,9mm

Pes: 135 grams

Velocitat de CPU: 1GHz

Antena GPS Interna

Pantalla

- Tipus: Pantalla Tàctil AMOLED
- Mida: 3,7 polzades
- Resolució: 480x480 WGA

Emmagatzematge:

- ROM: 512 MB
- RAM: 576 MB
- Targeta Micro SD 4GB

4.4.2 Eines Software

Eclipse SDK

Eclipse es un entorn de desenvolupament integrat, obert i extensible. Està composat per una sèrie d'eines per el fàcil desenvolupament de software. Disposa de eines tals que un editor de codi, un compilador/intèrpret i un depurador de codi.

Per poder utilitzar aquest IDE també es necessari disposar d'un SDK d'Android disponible gratuïta per Eclipse.

Figura 32: Eclipse IDE

Android SDK per Eclipse

Aquets SDK es el plugin oficial d'Android que ens ofereix les eines necessàries per desenvolupar una aplicació en Android des de el entorn d'Eclipse. En el SDK podrem trobar un emulador que també hem pogut utilitzar per la realització d'algunes proves.

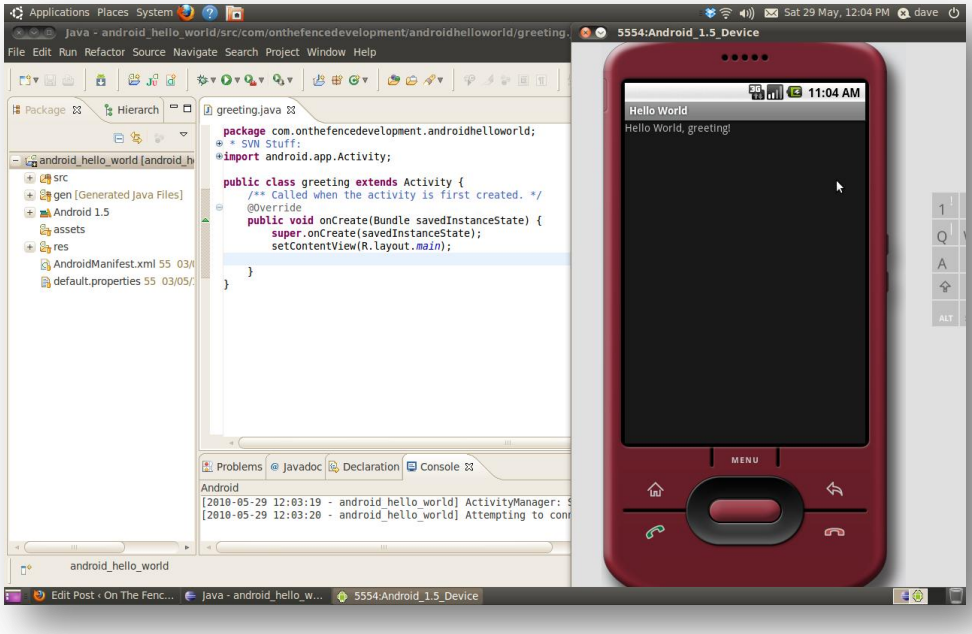


Figura 33: SDK Android per Eclipse

SQLite3

El SDK d'Android utilitzat ja porta inclòs la possibilitat de creació de base de dades amb SQLite. Aquesta es una base de dades relacional, a diferencia d'altres base de dades no es un procés autònom, sinó que s'integra dins d'altres programes.

Figura 34: SQlite per Android

5. Conclusions i futures ampliacions

En general la valoració de la realització del projecte ha estat bastant positiva, però hi ha coses que caldria destacar

Un dels aspectes mes positius ha sigut el de aprendre a utilitzar una tecnologia nova en el camp del desenvolupament d'aplicacions. D'aquesta manera he pogut adquirir nous coneixements que abans no tenia, i s'ha pogut millorar en la cerca de nous problemes de forma autònoma i en adquirir mes rapidesa en la forma de trobar-los.

A l'aplicació desenvolupada cal dir que es podria millorar de moltes maneres possibles, i que per raons de temps no s'han pogut realitzar, però crec que compleix les necessitats que ens havíem proposat des de el principi. Una de les possibles millores que es podria afegir seria la de connexió amb la xarxa social Facebook, una altra millora important que caldria afegir en un futur seria la de intentar connectar-se alguna xarxa de venta d'entrades per tal de poder aconseguir aquestes des de el nostre dispositiu. Per part de la interfície gràfica també seria bo una possible millora en el cas de que l'aplicació sortís algun dia al mercat.

6. Referencia Bibliogràfiques

- Android: Guia para desarrolladores (2ª Edición), Anaya Multimedia
- Stack-Overflow (http://stackoverflow.com/)
- Android Developers (http://developer.android.com/index.html)
- Last.fm (http://www.lastfm.es/api)
- API Twitter (https://dev.twitter.com/)
- API Google Maps (https://developers.google.com/maps/)
- Android-Spa (http://www.android-spa.com/)

7. Annexos

Els annexos de la memòria estan formats per parts de codi mencionats anteriorment en la part de desenvolupament de la memòria.

7.1 Instal·lació del projecte a Eclipse IDE

1. Instal·lar el plugin d'Android dins dek IDE Eclipse.
   http://developer.android.com/sdk/installing/installing-adt.html
2. Una vegada instal·lat correctament ens dirigim a File > New > Android Project.
3. A la finestra emergent, activar l'opció *create project with exising source.*
4. S'activa la barra d'opcions de sota. Busquem el projecte que es vol importar.
5. Seleccionem la versió d'Android. (Versió Google API 2.3.3)
6. Cliquem a Finish.

7.2 Instal·lació de l'aplicació

Per poder instal·lar en el dispositiu Android, es necessari executar el paquet instal·lador .apk que es troba adjunt dins del CD.

Es tindrà que copiar aquest paquet en la memòria del dispositiu i executar-lo per a que es pugui instal·lar amb facilitat.

7.3 Fragments de codi de l'aplicació

7.2.1 Pantalla principal. *TabLayoutActivity*.

```
<TabHost
xmlns:android="http://schemas.android.com/apk/res/android"android:id="@an
droid:id/tabhost"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
            <LinearLayout
                  android:orientation="vertical"
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent">
                  <TabWidget
                  android:id="@android:id/tabs"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content"
                  />
                  <FrameLayout
                  android:id="@android:id/tabcontent"
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent"
                  />
      </LinearLayout>
</TabHost>
```

Codi 7.1: TabLayout XML

```
TabHost tabHost = getTabHost();
//Song list Tab
TabSpec listTabSpec = tabHost.newTabSpec("List");

listTabSpec.setIndicator("General
List",getResources().getDrawable(R.drawable.icon_song_layo
ut));
Intent listIntent = new
Intent(this,TrackListActivity.class);
listTabSpec.setContent(listIntent);
tabHost.addTab(listTabSpec);
```

Codi 7.2: TabLayout Java

7.2.2 Pestanya Principal. Llista de cançons.

```
Cursor cur =
managedQuery(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,

columns,Audio.Media.DATA + " like ? OR " + Audio.Media.DATA
+ " like ? ",new
String[]{"%mp3"},MediaStore.Audio.Media.TITLE);
```

Codi 7.3: Extracció de música de la targeta SD

7.2.3 Reproductor musical.

```
void doBindService() {

getApplicationContext().bindService(serviceIntent,
serviceConnection,

Context.BIND_AUTO_CREATE);

Log.d("Bind", "do bind service");
```

Codi 7.4: Connexió activitat-servei

```
registerReceiver(broadcastReceiver, new IntentFilter(
MediaPlayerActivity.BROADCAST_SEEKBAR));

private BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {
      @Override
      public void onReceive(Context context, Intent intent) {

            updateSeekPos(intent);
      }
};
```

Codi 7.5: Registre i creació del Broadcast

```
private void initNotification() {

NotificationManager mNotificationManager = (NotificationManager)

getSystemService(Context.NOTIFICATION_SERVICE);

Intent intent = new Intent(getApplicationContext(),
MediaPlayerActivity.class);
intent.putExtra("noti", true);
intent.putExtra("notification", true);
intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP
| Intent.FLAG_ACTIVITY_SINGLE_TOP);
pending = PendingIntent.getActivity(getApplicationContext(), 0,
intent,0);
int icon = R.drawable.logoaplicacio;
CharSequence tickerText = "Media Player";
long when = System.currentTimeMillis();
noti = new Notification(icon, tickerText, when);
noti.flags = Notification.FLAG_NO_CLEAR;
noti.setLatestEventInfo(getApplicationContext(), "Media
Player","playing: " + trackNames.get(current), pending);

mNotificationManager.notify(NOTIFICATION_ID, noti);


}
```

Codi 7.6 Creació de la notificació

7.2.4 Localització d'esdeveniments musicals

```
Public CustomItemizedOverlay(Drawable
defaultMarker,Context context,int type) {

      this(defaultMarker);
      this.context = context;
      this.type = type;
}
```

Codi 7.7: Constructor CustoItemizedOverlay

```
protected boolean onTap(int index) {

OverlayItem item = mapOverlays.get(index);
if(type == 1) {
Intent intent = new
Intent(context,EventInfoActivity.class);
intent.putExtra("event",event);
context.startActivity(intent);
}

return true;

}
```

Codi 7.8: Mètode onTap d'un marcador

```
CustomItemizedOverlay item = new
CustomItemizedOverlay(draw, context, 1);

item.setEvent(events.get(i));
GeoPoint point = new GeoPoint(latE, lonE);
OverlayItem overlayItem = new OverlayItem(point,
eventsId.get(i).getArtist(),message);
item.addOverlay(overlayItem);
mapOverlays.add(item);
```

Codi 7.9: Creació d'un nou marcador

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

Codi 7.10: Obtenir instancia del servei de localització

```
LocationListener locListener = new LocationListener() {

@Override

public void onLocationChanged(Location location) {
// TODO Auto-generated method stub

      if (maps.isBetterLocation(lastLocation, location)) {
            betterLocation = location;
      } else {
            betterLocation = lastLocation;
      }

}

@Override
public void onProviderDisabled(String provider) {
// TODO Auto-generated method stub

}

@Override
public void onProviderEnabled(String provider) {
// TODO Auto-generated method stub

}

@Override
public void onStatusChanged(String provider, int status, Bundle extras)
{
// TODO Auto-generated method stub

}

};
locationManager.requestLocationUpdates(bestProvider, 0, 0,
locListener);
```

Codi 7.11: Listener d'actualitzacions de posició.

```
public boolean isBetterLocation(Location location,Location
currentBestLocation) {

      //If the current is null the better is the new location
      if (currentBestLocation == null) {
            return true;
      }
      //Checks the best by time.
      long timeDelta = location.getTime() -
      currentBestLocation.getTime();
      boolean isSignificantlyNewer = timeDelta > TWO_MINUTES;
      boolean isSignificantlyOlder = timeDelta < -TWO_MINUTES;
      boolean isNewer = timeDelta > 0;

      if (isSignificantlyNewer)
            return true;
      else if (isSignificantlyOlder)
            return false;

      //Checks the better by accuracy
      int accuracyDelta = (int) (location.getAccuracy() -
      currentBestLocation.getAccuracy());
      boolean isLessAccurate = accuracyDelta > 0;
      boolean isMoreAccurate = accuracyDelta < 0;
      boolean isSignificantlyLessAccurate = accuracyDelta >
      200;
      boolean isFromSameProvider =
      isSameProvider(location.getProvider(),
      currentBestLocation.getProvider());

      if (isMoreAccurate)
            return true;
      else if (isNewer && !isLessAccurate)
            return true;
      else if (isNewer && !isSignificantlyLessAccurate &&
      isFromSameProvider)
            return true;
      return false;

}
```

Codi 7.12: Cerca de la millor localització

```
public int getDistance(int latA, int lonA, int latB, int lonB)
{

	int Radius = 6371000; // Radio de la tierra
	double lat1 = latA / 1E6;
	double lat2 = latB / 1E6;
	double lon1 = lonA / 1E6;
	double lon2 = lonB / 1E6;
	double dLat = Math.toRadians(lat2 - lat1);
	double dLon = Math.toRadians(lon2 - lon1);
	double a = Math.sin(dLat / 2) * Math.sin(dLat / 2)
		+ Math.cos(Math.toRadians(lat1))
		* Math.cos(Math.toRadians(lat2))
		* Math.sin(dLon / 2)
		* Math.sin(dLon / 2);
	double c = 2 * Math.asin(Math.sqrt(a));
	return (int) (Radius * c);


}
```

Codi 7.13: Calcul de la distancia entre punts

7.2.5 Pantalla d'esdeveniment musical

```
ContentValues vals = new ContentValues();

vals.put("dtstart",startMilis);
vals.put("dtend",endMilis);
vals.put("title",event.getTitle());
vals.put("description",event.getArtist() + "\n" +
event.getPlace() + ", " +
event.getStreet() + ", " + event.getCity());
vals.put("calendar_id",idCal);
Uri u = Uri.parse("content://com.android.calendar/events");
Uri uri = activity.getContentResolver().insert(u, vals);
```

Codi 7.14: Inserció d'esdeveniments al calendari

```
ContentValues values = new ContentValues();

values.put("minutes",1440);
values.put("event_id",eventID);
values.put("method", 1);
Uri uRemind =
Uri.parse("content://com.android.calendar/reminders");
Uri ur = activity.getContentResolver().insert(uRemind,
values);
```

Codi 7.15: Inserció d'alertes al calendari

7.2.6 Connexió amb xarxes socials

```
this.consumer=new CommonsHttpOAuthConsumer(TwitterConstants.CONSUMER_KEY,

TwitterConstants.CONSUMER_SECRET);
this.provider = new CommonsHttpOAuthProvider(TwitterConstants.REQUEST_URL,
TwitterConstants.ACCESS_URL,TwitterConstants.AUTHORIZE_URL);
new TwitterRequestTokenTask(this,consumer,provider).execute();
```

Codi 7.16: Autenticació a Twitter

```
final String url = provider.retrieveRequestToken(consumer,
TwitterConstants.OAUTH_CALLBACK_URL);

Intent intent = new Intent(Intent.ACTION_VIEW,
Uri.parse(url)).setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP |
Intent.FLAG_ACTIVITY_NO_HISTORY | Intent.FLAG_FROM_BACKGROUND);
context.startActivity(intent);
```

Codi 7.17: Obertura del navegador de login

```
public void onNewIntent(Intent intent) {

      super.onNewIntent(intent);
      SharedPreferences prefs =
      PreferenceManager.getDefaultSharedPreferences(this);
      final Uri uri = intent.getData();
      if(uri != null &&
      uri.getScheme().equals(TwitterConstants.OAUTH_CALLBACK_SCHEM
      E)) {
      Log.i(TAG, "Callback received : " + uri);
      Log.i(TAG, "Retrieving Access Token");
      new
      RetrieveAccessTokenTask(this,consumer,provider,prefs).execut
      e(uri);
      finish();
}
```

Codi 7.18: Interceptar el callback

```
public static void sendTweet(SharedPreferences prefs,String msg)
throws TwitterException {

String token = prefs.getString(OAuth.OAUTH_TOKEN, "");
String secret = prefs.getString(OAuth.OAUTH_TOKEN_SECRET, "");
AccessToken a = new AccessToken(token,secret);
Twitter twitter = new TwitterFactory().getInstance();
twitter.setOAuthConsumer(TwitterConstants.CONSUMER_KEY,
TwitterConstants.CONSUMER_SECRET);
twitter.setOAuthAccessToken(a);

twitter.updateStatus(msg);

}
```

Codi 7.19: Enviar el nostre Tuït