UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea quinquennale in Informatica

PLAYER MUSICALE PER IPAD BASATO SULLA TEORIA DEI GRAFI

Relatore:

Dott. Luca Andrea Ludovico

Correlatore:

Dott. Adriano Baratè

Tesi di Laurea di

Daniele Addabbo

Matr. 529004

Anno Accademico 2015-2016

INDICE

Per Emer

Capitolo 1 - Introduzione

1.1 Descrizione della tesi

Le librerie musicali, anche personali, possono contenere migliaia di brani. Sebbene sia diventato comune, attraverso l'uso degli smartphone e dei tablet avere la possibilità di portare con sé migliaia di brani, la ricerca all'interno di queste banche dati rappresenta ancora un problema.

Spesso in queste librerie non è consentito fare ricerche che combinino informazioni e colleghino tra loro ad esempio gli artisti oppure i brani. Inoltre la rappresentazione dei risultati è spesso una semplice tabella.

La rappresentazione in tabella della libreria musicale sebbene molto semplice e intuitiva, a volte può rendere macchinosa la ricerca all'interno di banche dati musicali molto ricche, ad esempio con centinaia di brani per ciascun artista, specialmente sui piccoli schermi dei device mobili. Inoltre, la rappresentazione in tabella non consente di mettere facilmente in relazione brani appartenenti a diversi artisti e a diversi album.

Il presente lavoro di tesi intende realizzare una App per tablet (iPad di Apple) per la riproduzione musicale che permetta di navigare all'interno della propria libreria tramite un grafo che colleghi tra loro i brani, gli album e gli artisti.

Lo scopo di questa App è quello di dimostrare che è possibile rappresentare le librerie musicali usando forme alternative alla rappresentazione tabellare.

Si vuol dimostrare che la teoria dei grafi permette di realizzare una soluzione intuitiva, versatile, di semplice utilizzo ed efficace nel consentire all'utente di districarsi all'interno di migliaia di brani. Inoltre si intende dimostrare che la rappresentazione della libreria con un grafo consente di fornire all'utente nuovi ed innovativi collegamenti tra gli elementi della libreria che consentono di meglio apprezzare i brani contenuti nella libreria stessa. Questa App è nata come risposta ai limiti precedentemente descritti dell'interfaccia utente diffusa nei dispositivi in commercio.

Un semplice esempio di un collegamento all'interno della libreria musicale che non viene reso facilmente utilizzabile dalle interfacce utente attualmente diffuse è il collegamento tra brani con lo stesso titolo. Si pensi ad esempio alle cover ovvero alle reinterpretazioni di brani musicali da

parte di qualcuno che non ne è l'interprete originale. Supponiamo che un utente abbia selezionato un artista, quindi un album ed infine un brano musicale. Mentre l'utente ascolta questo brano sarebbe molto utile sapere se nella libreria musicale esistono una o più versioni differenti di quel brano.

Un altro esempio di collegamento che un utente potrebbe voler fare ma non può a causa dell'interfaccia grafica è quella tra due artisti. Supponiamo che un artista abbia iniziato la propria carriera in un gruppo musicale e che poi abbia intrapreso la carriera da solista, sarebbe interessante poter creare un collegamento tra queste due entità in modo che quando si ascolta la musica di uno dei due artisti l'interfaccia possa suggerire all'utente di ascoltare anche l'altro.

Con le interfacce utente attualmente più diffuse questo tipo di collegamento, ed altri ancora che verranno analizzati, non è consentito o risulta poco pratico.

1.2 Organizzazione della tesi

Il presente lavoro di tesi è organizzato in sette capitoli. Ogni capitolo è stato suddiviso in sotto capitoli per facilitare la lettura e per rendere più veloce la consultazione.

Il capitolo uno rappresenta una breve introduzione al presente lavoro di tesi.

Il capitolo due contiene un panoramica dello stato dell'arte dei player musicali software sia passati che presenti. Viene inoltre presentato un riepilogo dello stato dell'arte nell'ambito della ricerca sugli argomenti affini a questo lavoro di tesi. Questo capitolo serve a mostrare che nel mercato attuale tutte le implementazioni di player musicali più popolari si basano su una rappresentazione tabellare.

Nel capitolo tre vengono mostrate le principali tecnologie hardware e software che sono state usate durante il presente lavoro di tesi. Si parte dalla descrizione dell'iPad, per poi passare al suo sistema operativo iOS, all'ambiente di sviluppo integrato Xcode ed al linguaggio Objective-C.

Nel capitolo quattro vengono fornite le nozioni più importanti sulla Teoria dei Grafi, queste nozioni servono a comprendere il lavoro di tesi svolto. Viene inoltre data un'introduzione alle tecniche standard per la gestione in memoria dei grafi.

Nel capitolo cinque vengono mostrare le fasi che hanno portato alla realizzazione di questo lavoro di tesi. Si parte dalla creazione del progetto alla scelta delle tecnologie e librerie utilizzate. Vengono analizzate in dettaglio le librerie audio e le librerie grafiche che sono state utilizzate. Nel capitolo è presente anche la parte dedicata allo sviluppo del codice.

Nel capitolo sei vengono analizzati gli aspetti di layout e di funzionamento della App.

Nel capitolo sette vengono analizzate le conclusioni di questo lavoro di tesi e vengono proposti alcuni sviluppi possibili per l'evoluzione di quanto realizzato.

Capitolo 2 – Lo stato dell'arte

In questo capitolo viene presentato un elenco dei software più diffusi e rappresentativi per la riproduzione di musica dotati di interfaccia grafica nelle varie piattaforme disponibili sul mercato.

Viene inoltre presentato un riepilogo dello stato dell'arte nell'ambito della ricerca sugli argomenti affini a questo lavoro di tesi.

2.1 Desktop

In ambito desktop, i primi player musicali dotati di interfaccia grafica in grado di riprodurre file audio salvati all'interno del computer si sono diffusi all'inizio degli anni '90. Nel 1991 sono nati sia QuickTime della Apple [1] sia Media Player della Microsoft [2].

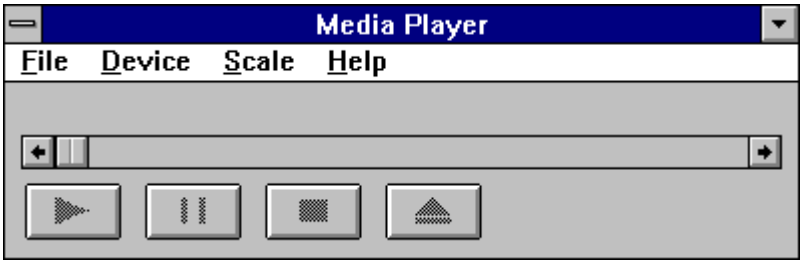


Fig. 1 Microsoft Media Player in Windows 3.1 (1992)

Fig. 2 Apple QuickTime in Mac OS 9 (1999)

A differenza dei player precedenti che erano in grado solo di comandare il lettore CD del computer, che poi si occupava di leggere direttamente in hardware il contenuto del CD Audio (Compact Disc Digital Audio), questi player erano già in grado di riprodurre file di diverso formato presenti sull'hard disk del computer. Gli hard disk degli anni '90, però, avevano una capacità di circa 500 MB (tipico computer del 1995) ed il formato wave (.wav) non compresso occupava troppo spazio (circa 10MB al minuto per un file in qualità CD). Questo rendeva impraticabile la creazione di una libreria musicale in formato audio salvata su hard disk.

Nel 1994 il Fraunhofer Institute [3] rilasciò al pubblico il primo software in grado di comprimere i file PCM (.wav) in file MP3 (MPEG-1 o MPEG-2 Audio Layer III), un formato audio che usa una compressione audio di tipo lossy. Il formato MP3 è in grado di ridurre drasticamente la quantità di dati richiesti per memorizzare un file audio, con una qualità audio comunque abbastanza fedele al file originale non compresso. Un file MP3 di buona qualità richiede da 1 a 2 MByte al minuto).

Il proliferare dei file MP3 e di altri formati simili come Advanced Audio Coding (AAC) ed allo stesso tempo l'aumento della capacità di archiviazione degli hard disk ha portato alla nascita di nuovi software, tra i quali Winamp della Nullsoft, Inc. (1997) [4].

Fig. 3 Nullsoft Winamp 3.7 (2003)

Questi nuovi software introducevano la possibilità di creare delle playlist, vale a dire elenchi di brani che venivano riprodotti in sequenza.

E' soltanto alcuni anni più tardi che, anche grazie alla diffusione degli MP3 Player hardware portatili, nascono i primi software in grado di gestire una libreria musicale vera e propria. Nel 2001 Apple rilascia la prima versione di iTunes (player musicale evoluto) [5] e la prima versione di iPod (lettore di musica digitale portatile).

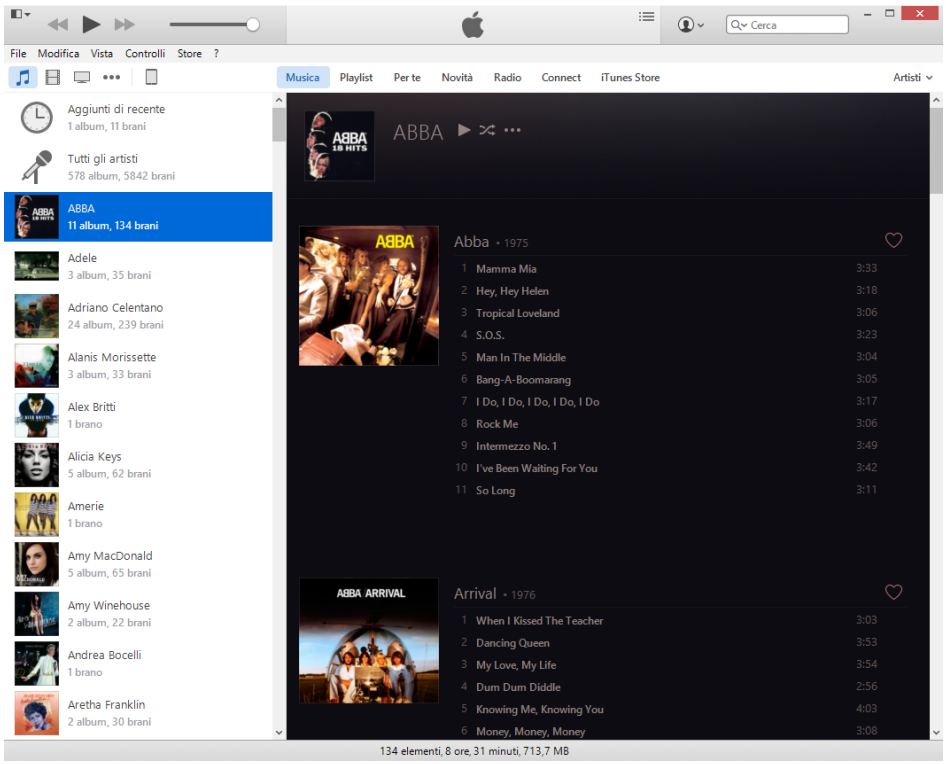


Fig. 4 Apple iTunes 12.3 (2015)

Questa nuova generazione di software consente di creare delle librerie musicali di notevoli dimensioni. Le librerie, sempre mostrare in forma tabellare, possono essere visualizzate secondo uno di questi ordinamenti: genere, artista, album, brano, compositore.

2.2 Smartphone

Gli smartphone dotati di touch screen, come l'iPhone di Apple, sono stati introdotti nel 2007 e negli ultimi anni sono diventati i media player portatili per eccellenza. Sono molto diffusi ed hanno schermi con una diagonale da 3,5" a 5,5", risoluzione dello schermo che spesso arriva a 1920x1080 pixel ed una memoria per l'archiviazione dei contenuti che arriva fino a 128GB.

Il software (App) per la riproduzione della musica negli iPhone della Apple [6] (e nella maggior parte degli altri smartphone) è organizzato in quattro schermate:

- Elenco artisti
- Elenco album di un artista
- Elenco brani di un album
- Pagina dedicata alla riproduzione del brano scelto

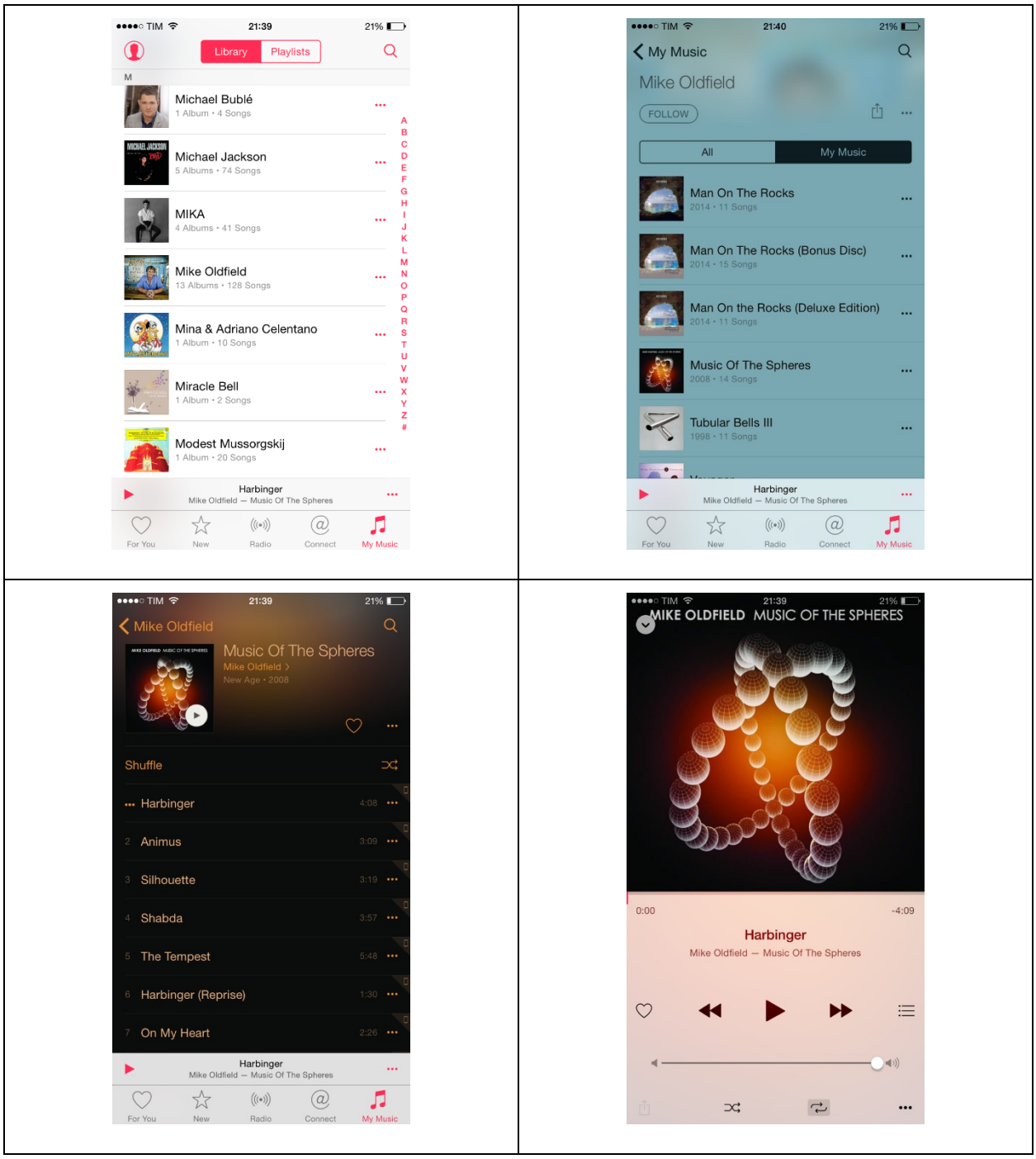


Fig. 5 App "Musica" in Apple iPhone con iOS 8.4.1 (2015)

Come si può chiaramente vedere l'interfaccia grafica è costituita da tabelle. Queste tabelle non consentono in alcun modo di mettere in evidenza possibili relazioni tra gli artisti, tra gli album o tra i brani.

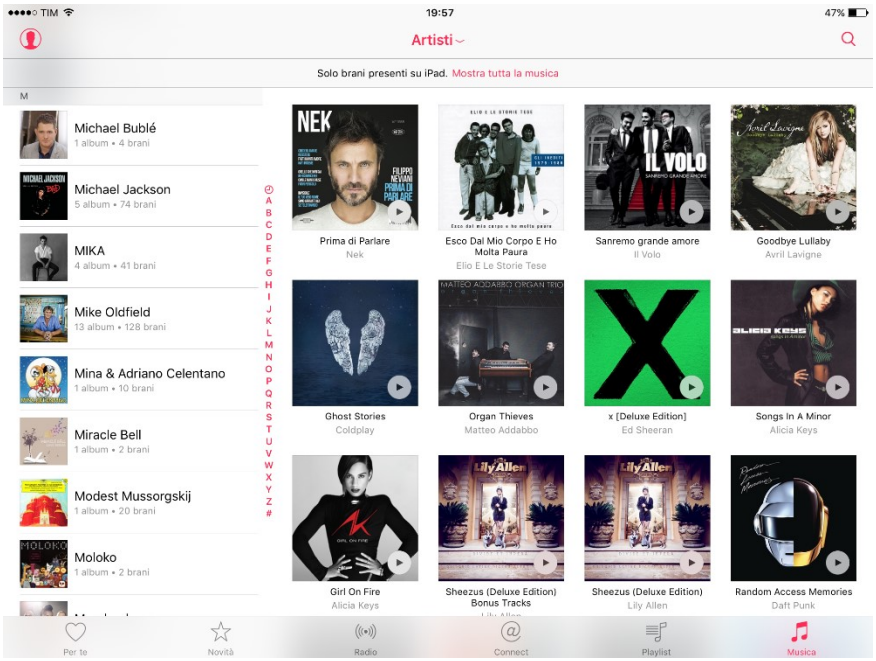
2.3 Tablet

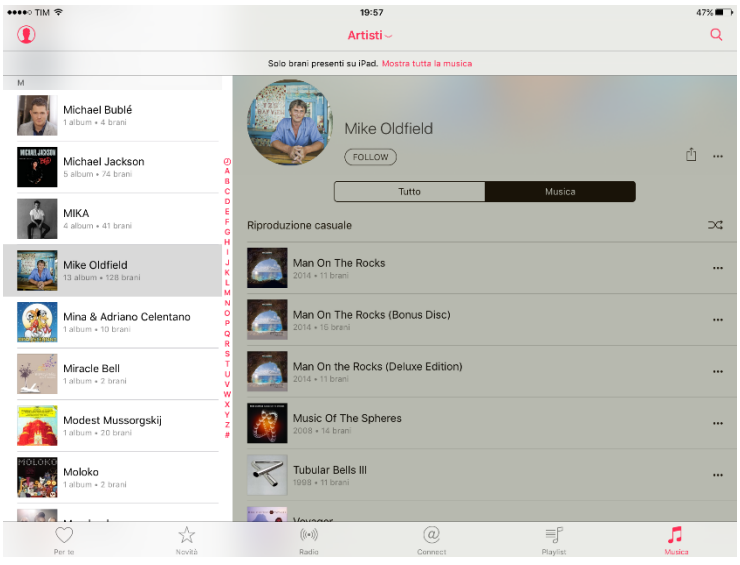
I tablet sono nati come anello di congiunzione tra gli smartphone ed i computer tradizionali. L'attuale generazione di tablet è nata nel 2010 con l'introduzione dell'iPad di Apple. I tablet sono

dotati di schermi con una diagonale da 7" a 13", risoluzione dello schermo che arriva a 2732×2048 pixel ed una memoria per l'archiviazione dei contenuti che arriva fino a 128GB.

Anche il software (App) per la riproduzione della musica negli iPad della Apple [7] (e nella maggior parte degli altri tablet) è organizzato in quattro schermate, concettualmente molto simili a quelle già viste su iPhone:

- Elenco artisti
- Elenco album di un artista
- Elenco brani di un album
- Pagina dedicata alla riproduzione del brano scelto





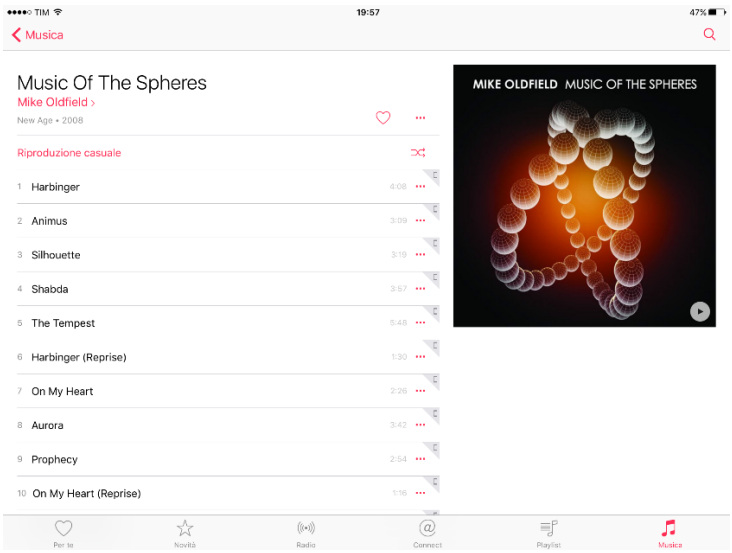
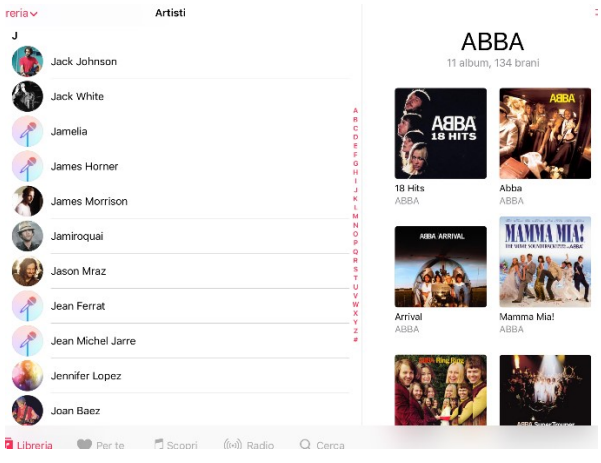


Fig. 6 App "Musica" in Apple iPad con iOS 9.0.2 (2015)

Quelle che seguono sono le schermate dell'ultima versione di iOS per iPad attualmente disponibile.



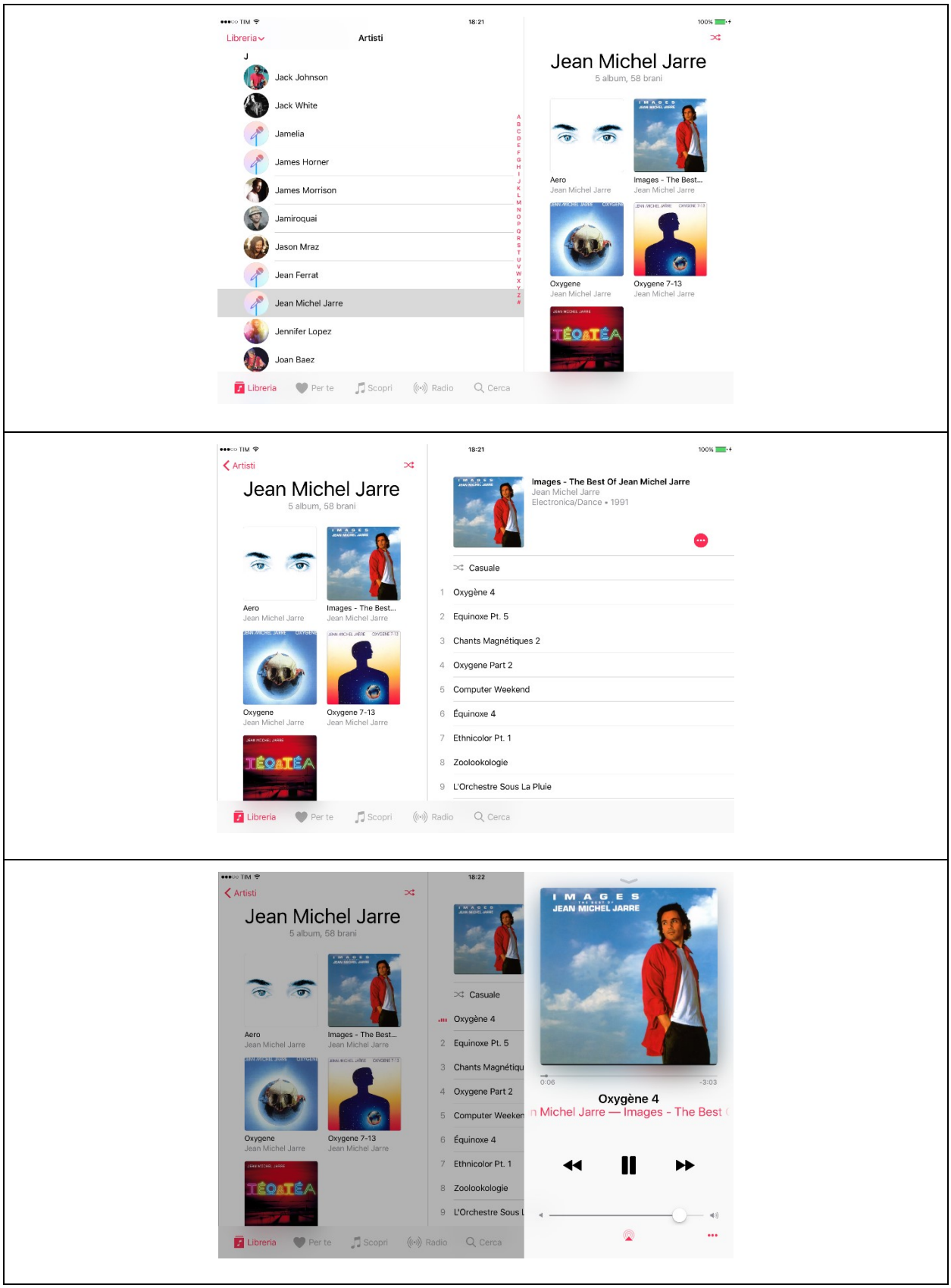


Fig. 7 App "Musica" in Apple iPad con iOS 10.1.1 (2016)

Come si può vedere, anche nel caso dell'iPad l'interfaccia grafica è costituita da tabelle. Queste tabelle non consentono in alcun modo di mettere in evidenza possibili relazioni tra gli artisti, tra

gli album o tra i brani. Inoltre queste schermate non sfruttano lo schermo di maggiori dimensioni di cui è dotato l'iPad: il contenuto delle schermate è molto simile a quello dell'iPhone.

2.4 Google Wonder Wheel

Google, il motore di ricerca più diffuso in internet, è costantemente alla ricerca di nuove tecnologie in grado di migliorare il proprio prodotto.

Tra i progetti di Google ci sono nuove opzioni dedicate alla rappresentazione nei risultati delle ricerche, chiamate Google Search Options. Nel 2009 Google ha sperimentato un sistema di rappresentazione dei risultati tramite l'uso dei grafi. L'opzione si chiamava "Wonder Wheel". Il progetto Wonder Wheel è stato abbandonato da Google. Anche se non si tratta di un software ottimizzato per la riproduzione di musica, riteniamo utile inserire un nota dedicata a questo software per evidenziare che anche aziende importanti come Google hanno ritenuto utile investire nella ricerca di una rappresentazione grafica diversa dalle tabelle.

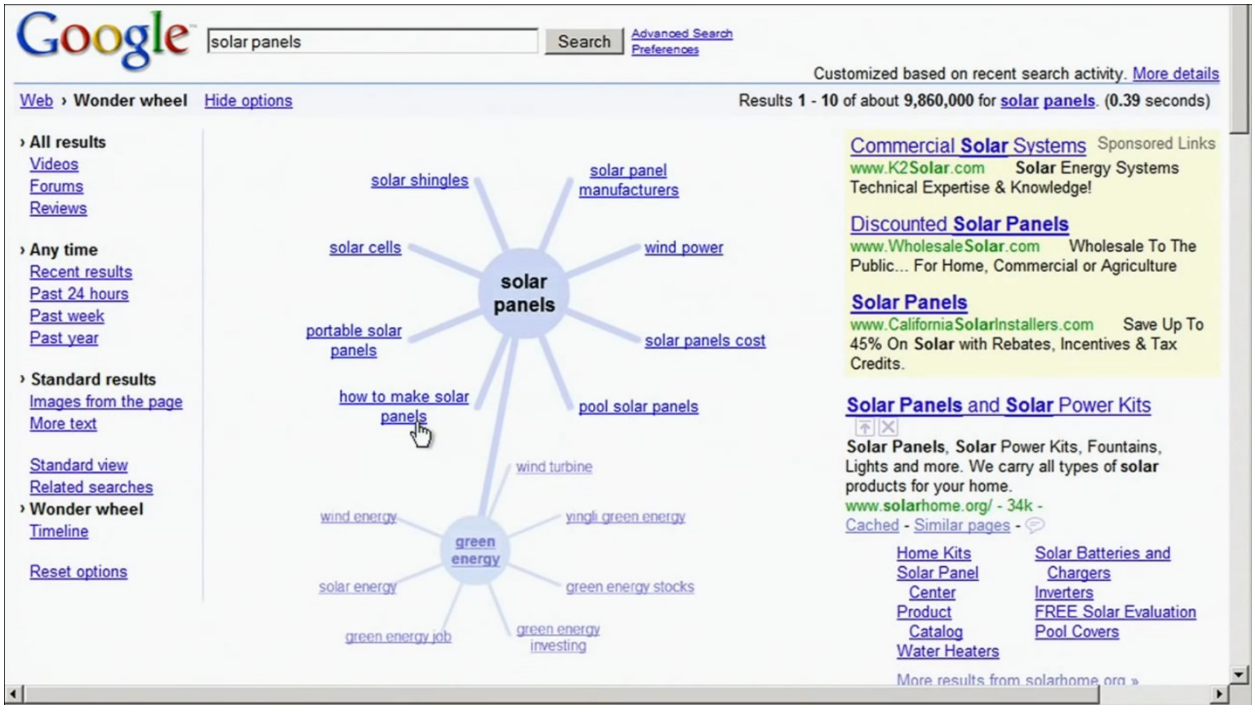


Fig. 8 Google Wonder Wheel

2.5 Music Mood Wheel

Music Mood Wheel [8] è un progetto realizzato tra il 2005 ed il 2006 dal dipartimento di Informatica dell'Università degli Studi di Milano e dalla Microsoft Research Cambridge.

Il progetto si basa sull'idea di creare un approccio diverso alla ricerca della musica che si desidera ascoltare. Invece di usare la ricerca testuale si usa una ricerca basata solo sul suono. Partendo dall'idea della manopola delle vecchio radio analogiche, il progetto usa la trackball di un computer per effettuare le ricerche, permettendo all'utente, senza l'uso di uno schermo, di muoversi in uno spazio bidimensionale sul quale sono state distribuiti i brani presenti nella libreria musicale. Gli elementi presi in considerazione per distribuire i brani all'interno dello spazio 2D sono ad esempio l'intensità di suono, i battiti al minuto ed il volume medio dei brani musicali. Questi elementi sono stati usati a coppie (uno per asse) in diversi esperimenti.

2.6 Generazione automatica di playlist

La generazione di playlist è un argomento di ricerca vivo soprattutto da quando la riproduzione di musica è passata dai CD-Audio ai media player.

La generazione manuale di playlist richiede, a volte, molto tempo per questo sono stati sviluppati diversi algoritmi per la generazione automatica di playlist. Ad esempio citiamo il lavoro "Automatic playlist generation based on tracking user's listening habits" di A. Andric e G. Haus del 2006 [9].

Ci sono principalmente due tipi di algoritmi per la generazione automatica di playlist, la differenza sta nel tipo di interazione con l'utente. Il primo tipo chiede all'utente di specificare uno o più brani che verranno poi usati per generare la playlist, che possiamo definire basata su un suggerimento. Il secondo tipo di algoritmo richiede invece all'utente di definire dei vincoli o limiti. Queste sono le playlist basate su vincoli. Mentre nel primo tipo di algoritmo l'utente esprime le proprie preferenze musicali tramite esempi, nel secondo tipo viene chiesto all'utente di esprimere in modo esplicito il tipo di contenuto musicale che vuole ascoltare.

Valutare la qualità delle playlist generate è un altro argomento molto interessante. A tal proposito citiamo il lavoro di A. Andric e G. Haus del 2005 [10] che mostra come valutare la qualità di una playlist è in generale molto difficile e richiede che le condizioni dell'esperimento siano preparate con attenzione. Nell'esperimento di Andric e Haus si è, ad esempio, ottenuto che

playlist generate manualmente dagli utenti hanno ottenuto punteggi simili alle playlist generate casualmente.

Nel generare e valutare le playlist è importante definire cosa si intende per gusto musicale e cosa invece si intente per preferenze musicali. Nel lavoro PATS (Personalized Automatic Track Selection), ad esempio, Pauws ed Eggen [11] seguono le definizioni già introdotte da Abeles [12]. Il gusto musicale viene definito come un legame a lungo termine che una persona crea lentamente verso un particolare stile musicale, il suo sviluppo dipende dall'ambiente culturale, dal consenso popolare, dal gusto dei coetanei, dalla personale cultura musicale, dall'età e da altre caratteristiche personali. D'altra parte la preferenza musicale è definita come il desiderio di ascoltare un particolare tipo di musica in uno specifico contesto, si tratta quindi di una preferenza soggetta a variare rapidamente nel tempo e che comunque dipende dal gusto musicale della persona.

Capitolo 3 - Tecnologie utilizzate

In questo capitolo vengono presentati gli strumenti hardware e software utilizzati per il presente lavoro di tesi.

3.1 Apple iPad

La piattaforma hardware che si è scelto di utilizzare è il tablet Apple iPad [7].

Si è scelto di utilizzare un tablet invece di un computer desktop perché la musica ben si presta ad essere fruita in mobilità. Ad esempio è possibile ascoltare la musica dell'iPad mentre si cammina (tramite le cuffie), in automobile (tramite il collegamento Bluetooth all'impianto stereo dell'auto), in qualsiasi stanza della casa (tramite la connessione WiFi all'impianto stereo di casa oppure tramite l'altoparlante integrato).

Si è scelto di utilizzare un tablet invece di uno smartphone perché lo schermo di maggiori dimensioni (9,7" invece di 5" circa) meglio si presta ad una rappresentazione grafica della libreria musicale.

Tra i vari modelli di tablet si è scelto di usare iPad di Apple perché è stato il primo tablet di nuova generazione ed è attualmente uno dei tablet più utilizzati. Da uno studio di IDC International Data Corporation [13], risulta che nel 2014 le vendite mondiali di iPad hanno rappresentato il 27,6% del mercato dei tablet e dei computer portatili 2 in 1. Secondo StatCounter, azienda specializzata nell'analisi del traffico web, a settembre 2015 il traffico web generato da iPad rappresenta il 66,04% del traffico generato dai tablet [14].

iPad è attualmente venduto in tre diverse versioni (viene mostrato solo l'ultimo modello per ciascuna dimensione):

	Diagonale dello schermo	Risoluzione (pixel)	Peso (grammi)
iPad Air 2	9,7" (246,4 mm)	2048×1536	444
iPad Mini 4	7,9" (200,7 mm)	2048×1536	300
iPad Pro	12,9" (327,7 mm) 9,7" (246,4 mm)	2732×2048 2048×1536	723 444

Si è scelto di utilizzare il modello iPad Air 2 da 9,7" in quanto ha una risoluzione adeguata (superiore al formato Full-HD), un hardware con ottime prestazioni ed uno schermo non troppo piccolo. Il peso, inoltre, lo rende adatto ad essere impugnato anche per lunghi periodi.

3.2 iOS

iOS [15] è un sistema operativo per dispositivi mobili creato e sviluppato da Apple ed è disponibile solo per dispositivi Apple. Si tratta di un sistema operativo Unix-like basato su Darwin (una versione open source di Unix creata da Apple e basata su BSD) ed OS X. E' stato lanciato nel 2007 insieme all'iPhone. Originariamente si chiamava iPhone OS perché era nato come sistema operativo per iPhone. Successivamente è stato ampliato per poter essere installato su altre periferiche come iPod Touch, iPad, iPad mini e iPad Pro.

iOS presenta una interfaccia grafica basata su un insieme di API denominate Cocoa Touch, derivate in parte da quelle di OS X chiamate Cocoa e ottimizzate per l'hardware basato su interfaccia touch screen [16].

Tabella 1 Timeline del sistema operativo iOS

Versione	Nome	Data di Rilascio
1	iPhone OS 1	29 Giugno 2007
2	iPhone OS 2	11 Luglio 2008
3	iPhone OS 3	17 Giugno 2009
4	iOS 4	21 Giugno 2010
5	iOS 5	12 Ottobre 2011
6	iOS 6	19 Settembre 2012
7	iOS 7	18 Settembre 2013
8	iOS 8	17 Settembre 2014
9	iOS 9	16 Settembre 2015
10	iOS 10	13 Settembre 2016

Al momento del lancio di iOS, il sistema operativo non supportava applicazioni di terze parti. Con il rilascio di iOS 2.0 è stato lanciato anche Apple App Store che ha consentito agli sviluppatori di terze parti di pubblicare le proprie applicazioni per iOS. La prima versione di iOS a supportare gli iPad è stata la versione 3.2.

L'ultima versione principale, la numero 10.x, è del Settembre 2016. L'ultimo aggiornamento di iOS, il 10.1.1, è di Ottobre 2016. Di solito la maggior parte dei dispositivi Apple viene aggiornata molto rapidamente all'ultima versione del sistema operativo. Ad esempio, dopo sole 3 settimane dal lancio il 57% dei dispositivi iOS erano stati aggiornati ad iOS 9 [17].

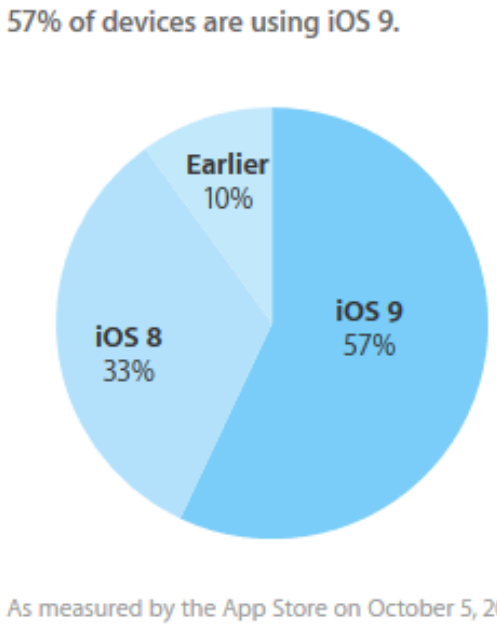


Fig. 9 Diffusione di iOS 9 a tre settimane dal lancio

Ad Agosto 2016, ad undici mesi dal lancio di iOS 9, l'88% dei dispositivi iOS erano stati aggiornati ad iOS 9 [17].

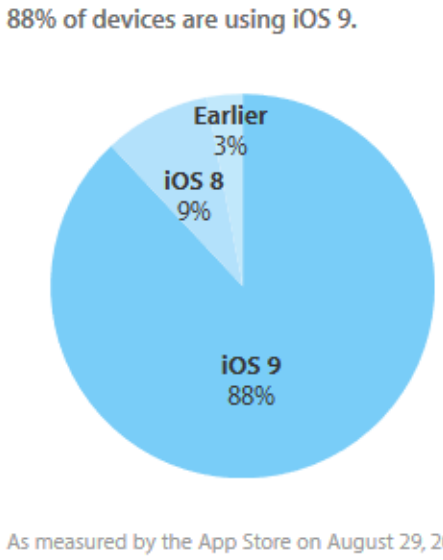


Fig. 10 Diffusione di iOS 9 ad unici mesi dal lancio

I dati più recenti ad oggi disponibili ci dicono che a circa due mesi dal lancio di iOS 10, il 63% dei dispositivi iOS sono stati aggiornati ad iOS 10.

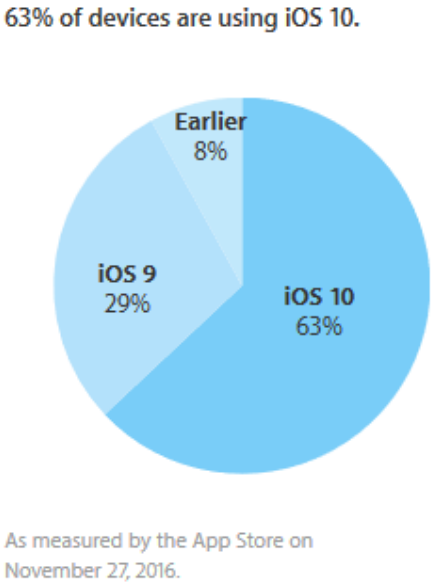


Fig. 11 Diffusione di iOS 10 a circa 2 mesi dal lancio

Ogni nuova release di iOS contiene nuove librerie ed ottimizzazioni importanti delle librerie esistenti, per questo motivo è consigliabile scrivere App pensate per l'ultima versione di iOS. Il fatto che l'ultima versione di iOS si diffonda molto velocemente è importante dal punto di vista degli sviluppatori che possono decidere di rilasciare App ottimizzate per la versione più recente del sistema operativo sapendo che la maggior parte degli utenti potranno installarle.

3.3 Apple Xcode

Xcode [18] è un ambiente di sviluppo integrato (Integrated development environment, IDE) contenente una suite di strumenti utili allo sviluppo di software per i sistemi OS X e iOS. Viene sviluppato da Apple. La prima versione di Xcode risale al 2003, la versione attuale, la numero 8.x, è del 2016 e consente di sviluppare applicazioni per dispositivi dotati dei seguenti sistemi operativi: iOS, watchOS, tvOS, and OS X. La famiglie di dispositivi per i quali è possibile sviluppare applicazioni con Xcode sono: iPhone, iPad, and Mac Apple TV, Apple Watch.

Xcode supporta diversi linguaggi di programmazione. Per lo sviluppo di applicazioni per iOS i due linguaggi più usati e supportati da Apple sono Objective-C e Swift.

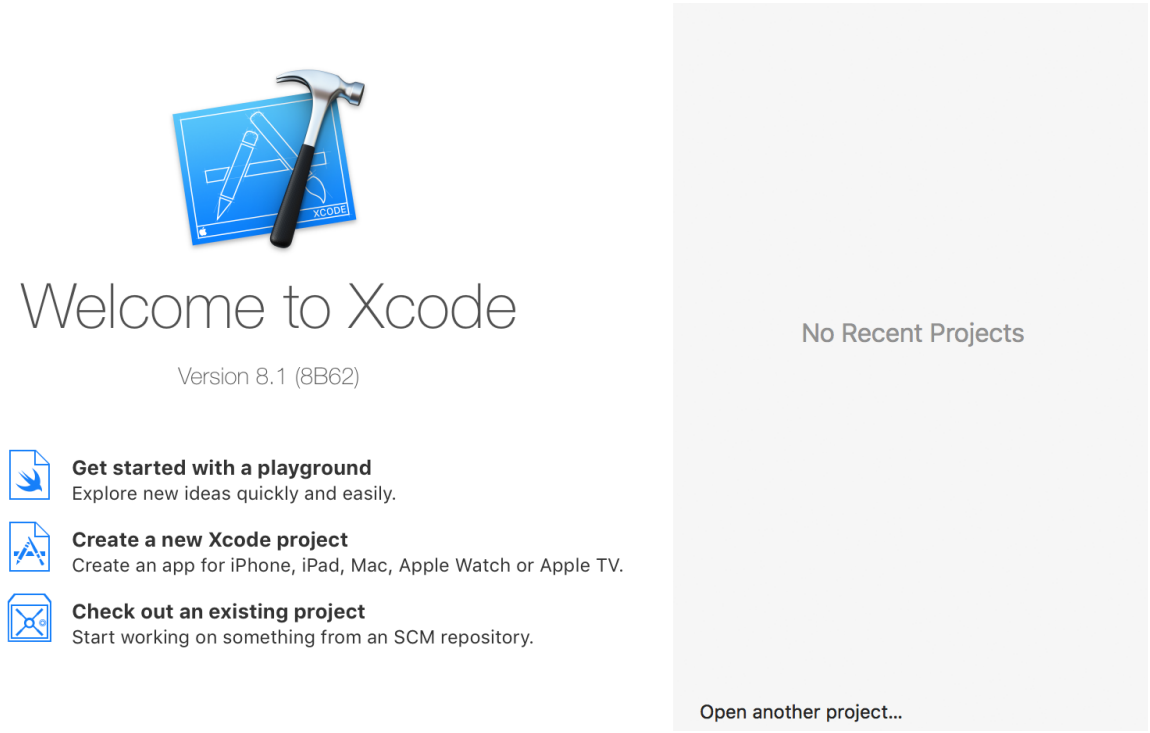


Fig. 12 Schermata di benvenuto di Xcode 8.1

Xcode comprende un simulatore iOS, chiamato iOS Simulator, che consente di testare le applicazioni che si stanno sviluppando senza essere costretti ad usare un device fisico. Il simulatore ha delle limitazioni, ad esempio non consente di simulare l'uso del microfono e della fotocamera. E' quindi consigliabile testare le applicazioni con un device reale prima di pubblicarle.

3.4 Objective-C

Objective-C è il principale linguaggio di programmazione usato per sviluppare applicazioni per piattaforme OS X e iOS di Apple.

Objective-C è un linguaggio di programmazione object-oriented sviluppato a partire dal 1983 da Brad Cox e Tom Love, i fondatori della società PPI (poi chiamata Stepstone). Objective-C è stato sviluppato partendo dal linguaggio C aggiungendovi le nuove capacità orientate agli oggetti introdotte dal linguaggio Smalltalk [16].

Nel 1985 Steve Jobs, dopo aver lasciato la Apple, fonda la società NeXT. Nel 1988 NeXT acquista la licenza dell'Objective-C dalla Stepstone e realizza un proprio compilatore Objective-C e nuove librerie per sviluppare l'interfaccia utente di quello che sarebbe stato il NeXT Operating System, chiamato NeXTSTEP.

Nel 1996 Apple acquista NeXT insieme alla licenza dell'Objective-C e al suo sistema di sviluppo Project Builder (in seguito rinominato Xcode) e con questi strumenti inizia a sviluppare il sistema operativo OS X.

Nel 2007 Apple presenta Objective-C 2.0, aggiungendo nuove proprietà al linguaggio e lo utilizza per creare il primo iPhone OS, poi chiamato iOS.

Objective C è un'estensione a oggetti del linguaggio C e mantiene la completa compatibilità col C. E' quindi possibile compilare un programma C con un compilatore Objective-C ed è possibile includere codice in linguaggio C all'interno di una classe Objective-C.

Queste sono alcune delle principali caratteristiche del linguaggio Objective-C [19]:

- Programmazione orientata agli oggetti (OOP): Objective-C fornisce un supporto completo alla OOP. Include caratteristiche come la comunicazione a scambio di messaggi, l'incapsulazione, l'ereditarietà, il polimorfismo.
- Comunicazione a scambio di messaggi: questa caratteristica permette agli oggetti di collaborare scambiandosi messaggi. Quello che accade è che un metodo o una funzione di una classe/oggetto manda un messaggio all'oggetto ricevente il quale usa il messaggio per invocare il metodo corrispondente, restituendo un valore se richiesto.
- Runtime dinamico: le caratteristiche del runtime system collocano l'Objective-C tra i linguaggi ad oggetti dinamici

- Gestione delle memoria: Objective-C fornisce una funzione automatica di gestione della memoria chiamata Automatic Reference Counting (ARC) che include un sistema di garbage collectors.
- Introspezione e riflessione: Objective-C consente di interrogare un oggetto a runtime e di modificare la sua struttura ed il suo comportamento.
- Supporto al linguaggio C: Objective-C è un'estensione orientata agli oggetti del linguaggio C. E' un sovrainsieme del C.

Apple fornisce diverse librerie e strumenti per lo sviluppo di applicazioni in Objective-C.

L'ultima versione del linguaggio Objective-C è la versione 2.0. Il compilatore usato da Xcode per il linguaggio Objective-C ed altri linguaggi è Apple LLVM. L'ultima versione del compilatore è la 8.0. In generale, le modifiche al compilatore LLVM introducono novità che ottimizzano anche il modo di scrivere il codice.

Capitolo 4 – Teoria dei grafi

4.1 Introduzione

I grafi sono una delle strutture più flessibili in cui organizzare i dati per favorire la gestione delle informazioni [20].

I grafi sono uno strumento di rappresentazione e modellazione di problemi. Come vedremo, la soluzione di molti problemi può essere ricondotta alla soluzione di opportuni problemi su grafi.

Verrà fornita la definizione classica di grafo. Per una trattazione più completa e rigorosa rimandiamo alla bibliografia dedicata all'argomento, ad esempio [21], [22].

4.2 Definizione

Dato un insieme V, denoteremo con V^s l'insieme delle s-ple ordinate di elementi di V e con $V^{(s)}$ la famiglia dei sottoinsiemi di V contenenti s elementi.

Ad esempio, se l'insieme *V={1, 2, 3}* allora:

$\{1, 2, 3\}^2 = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)\}$

$\{1, 2, 3\}^{(2)} = \{\{1,2\}, \{1,3\}, \{2,3\}\}$

Se V contiene n elementi, V^s ne contiene n^s, mentre $V^{(s)}$ ne contiene $\binom{n}{s} = \frac{n!}{s!(n-s)!}$

Definizione: Un **grafo** G è una coppia ordinata di insiemi disgiunti (V, E) dove E è un sottoinsieme dell'insieme $V^{(2)}$ delle coppie non ordinate di V.

Noi considereremo solo i grafi finiti, vale a dire che gli insiemi V ed E saranno sempre finiti.

- V è l'insieme dei nodi (o vertici)
- E è l'insieme dei lati (o archi)

Se G è un grafo, allora *V=V(G)* è l'insieme dei nodi di G mentre *E=E(G)* è l'insieme dei lati (o archi) di G.

Un lato (o arco) che unisce i nodi x ed y viene indicato con *xy* oppure *(x, y)*

Esistono due tipi di grafi: i grafi orientati ed i grafi non orientati.

Definizione: Un **grafo orientato** (o diretto) G è una coppia *G=(V, E)*, dove V è un insieme (finito) i cui elementi sono detti vertici (o nodi) ed E è un sottoinsieme di V^2 i cui elementi sono detti archi. Gli archi del tipo *(x, x)* sono detti cappi.

Definizione: Un **grafo non orientato** G è una coppia *G=(V, E)*, dove V è un insieme (finito) i cui elementi sono detti nodi (o vertici) ed E un sottoinsieme di $V^{(2)}$, i cui elementi sono detti lati (o archi).

I grafi non orientati non possiedono cappi.

Un sottografo di un grafo *G=(V, E)* è un grafo $G' = (V', E')$ tale che $V' \subseteq V$ e $E' \subseteq E$.

Se $xy \in E(G)$ allora x ed y sono nodi adiacenti di G ed i nodi x ed y sono incidenti con il lato *xy*. L'adiacenza del nodo x in un grafo $G = (V, E)$ è l'insieme dei nodi y tali che $(x, y) \in E$, cioè:

$$Adiacenza(x) = \{y | (x, y) \in E\}$$

Dato un lato *(x, y)* di un grafo orientato, diremo che x è la coda dell'arco ed y è la testa.

4.3 Rappresentazione grafica di un grafo

Come la terminologia stessa suggerisce, solitamente non si pensa ad un grafo in termini di coppie di nodi, ma come ad un insieme di nodi, alcuni dei quali sono collegati da lati. A volte il modo più semplice di descrivere un piccolo grafo è quello di disegnarlo.

La figura seguente dà una rappresentazione grafica del grafo orientato:

G_1 = ({1, 2, 3, 4, 5}, {(1,2), (2,2), (2,3), (3,1), (3,2), (4, 5), (5, 4)})

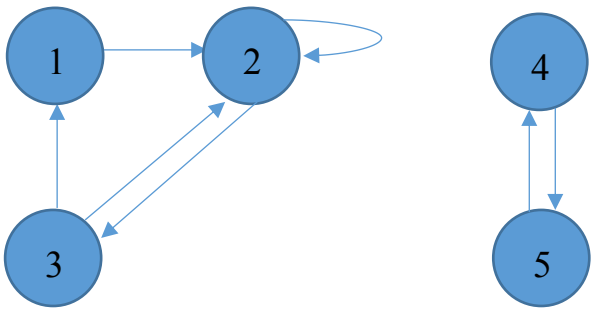


Fig. 13 Rappresentazione del grafo orientato G_1

La figura seguente dà una rappresentazione grafica del grafo non orientato:

$G_2 = (\{1, 2, 3, 4, 5\}, \{\{1, 2\}, \{2, 3\}, \{1, 3\}, \{4, 5\}\}$

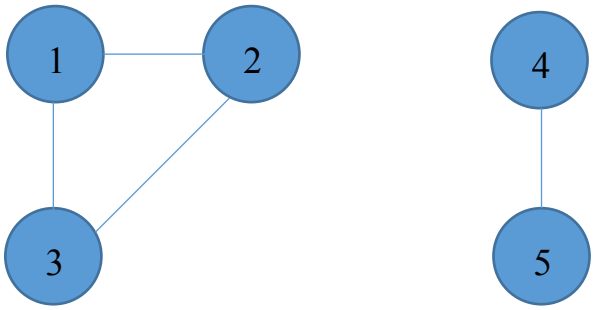


Fig. 14 Rappresentazione del grafo non orientato G_2

4.4 Proprietà

La teoria dei grafi è un campo estremamente vasto della matematica, in questa sezione illustreremo solo alcune proprietà dei grafi.

Ordine: si definisce ordine di un grafo il numero dei suoi nodi (o vertici)

Dimensione: la dimensione di un grafo è il numero dei suoi lati (o archi)

Cammino (o percorso): un cammino da x a y in un grafo $G=(V, E)$ è una sequenza $(v_1, v_2, \dots, v_m)$ di vertici tale che $v_1 = x$, $v_m = y$ e (v_k, v_{k+1}) è un arco di G, per tutti i $k = 1,2, \dots, m-1$. La lunghezza di questo cammino è $m-1$.

Un cammino si definisce semplice se tutti i suoi vertici sono distinti, eccetto al più il primo e l'ultimo.

Ciclo: si definisce ciclo un cammino semplice in cui il primo e l'ultimo vertice coincidono.

Le precedenti nozioni si estendono ai grafi non orientati. In questo caso i cicli devono avere lunghezza almeno 3.

Grafo connesso: un grafo si definisce connesso quando per ogni coppia x e y di vertici esiste un cammino da x a y.

Un grafo non orientato $G=(V, E)$ può non essere connesso, è però sempre esprimibile come l'unione di componenti connesse.

4.5 Grafi in informatica

Un grafo *G=(V, E)* può essere rappresentato dalla famiglia delle adiacenze di tutti i suoi vertici, cioè: $G = \{(x, Adiacenza(x)) | x \in V\}$

Ad esempio il grafo G_1 visto nel capitolo 4.3 può essere rappresentato con le adiacenze in questo modo: $\{(1,\{2\}), (2,\{2,3\}), (3,\{2,1\}), (4,\{5\}), (5,\{4\})\}$

In alternativa, un grafo G può essere descritto anche dalla sua matrice di adiacenza, vale a dire la matrice A_G indiciata con i vertici del grafo e a componenti in $\{0,1\}$, tale che:

$$A_G[x,y] = \begin{cases} 1 \; se \; (x,y) \in E \\ 0 \; altrimenti \end{cases}$$

Per esempio, questa è la matrice di adiacenza del grato G_1 visto nel capitolo 4.3:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Se un grafo G ha n vertici, l'implementazione della matrice di adiacenza richiederà un spazio di memoria $O(n^2)$.

Se il grafo G possiede un numero di lati notevolmente minore di n^2, può essere conveniente rappresentare il grafo come una lista di vertici, ognuno dei quali punta alla lista della sua adiacenza. Se il grafo G ha n nodi ed e lati, questa rappresentazione occuperà una memoria $O(n + e)$.

Alcuni moderni linguaggi di programmazione, come Objective-C, dispongono di tipi di dato ottimizzati per la rappresentazione dei grafi (si veda il capitolo 5).

Capitolo 5 – Realizzazione del player musicale

5.1 Introduzione

Per la realizzazione del player musicale oggetto di questo lavoro di tesi è stato necessario affrontare diverse fasi dello sviluppo. Innanzitutto è stato necessario configurare l'ambiente di sviluppo Xcode. Successivamente è stato necessario trovare il modo più efficace per accedere alla libreria musicale contenuta nell'iPad. In seguito si è affrontato il problema delle librerie grafiche da utilizzare per la rappresentazione dei grafi sullo schermo. Parallelamente è stato definito il Concept Layout (disposizione dei contenuti) per schematizzare e realizzare le schermate.

5.2 Creazione del progetto in Xcode

La prima fase della creazione di un progetto di Xcode parte della schermata che consente di selezionare il sistema operativo per il quale si vuole sviluppare (in questo caso iOS) ed uno dei modelli (template) previsti da Apple. Questi modelli sono per lo più simili tra loro e differiscono spesso solo per il tipo di impostazione grafica. E' comunque possibile modificare successivamente il progetto per includere gli elementi che si desidera inserire oppure per eliminare quanto non necessario [23]. Il modello che è stato scelto per questo lavoro di tesi si chiama "Single View Application".

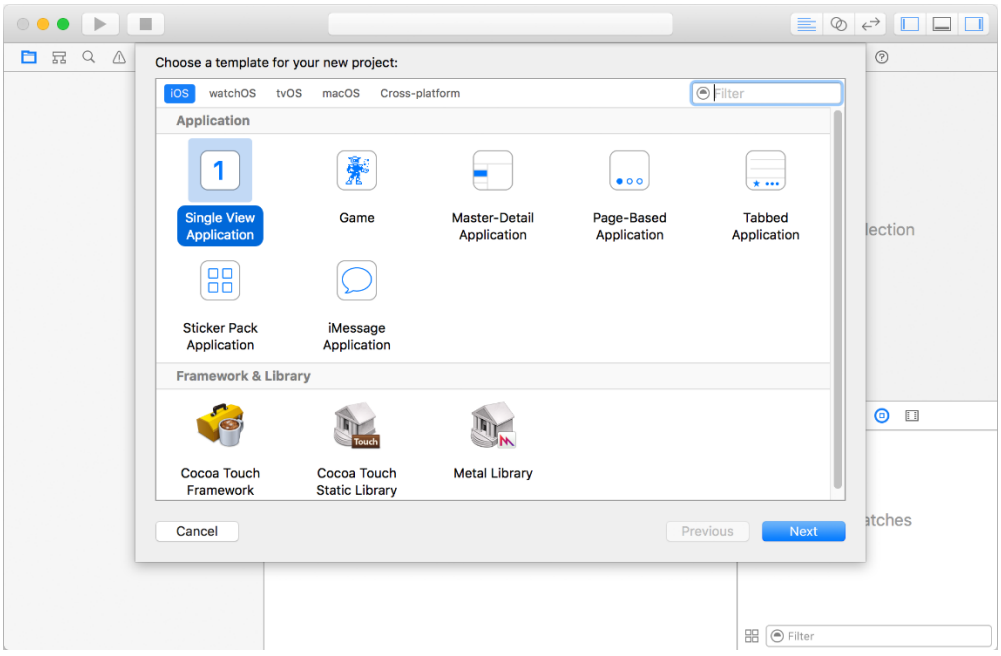


Fig. 15 Schermata di Xcode 8 per la scelta del sistema operativo e del modello

La schermata successiva richiede di specificare alcune informazioni indispensabili quali il nome dell'applicazione (Product Name) e l'organization identifier che è una sigla che identifica chi sta sviluppando l'applicazione. Questa schermata consente di effettuare due importanti scelte: il linguaggio di programmazione che si desidera utilizzare ed il tipo di dispositivo (device) per il quale si vuole sviluppare. Per questo progetto il tipo di dispositivo scelto è iPad.

Il nome scelto per questa App è "Music Graph AZ".

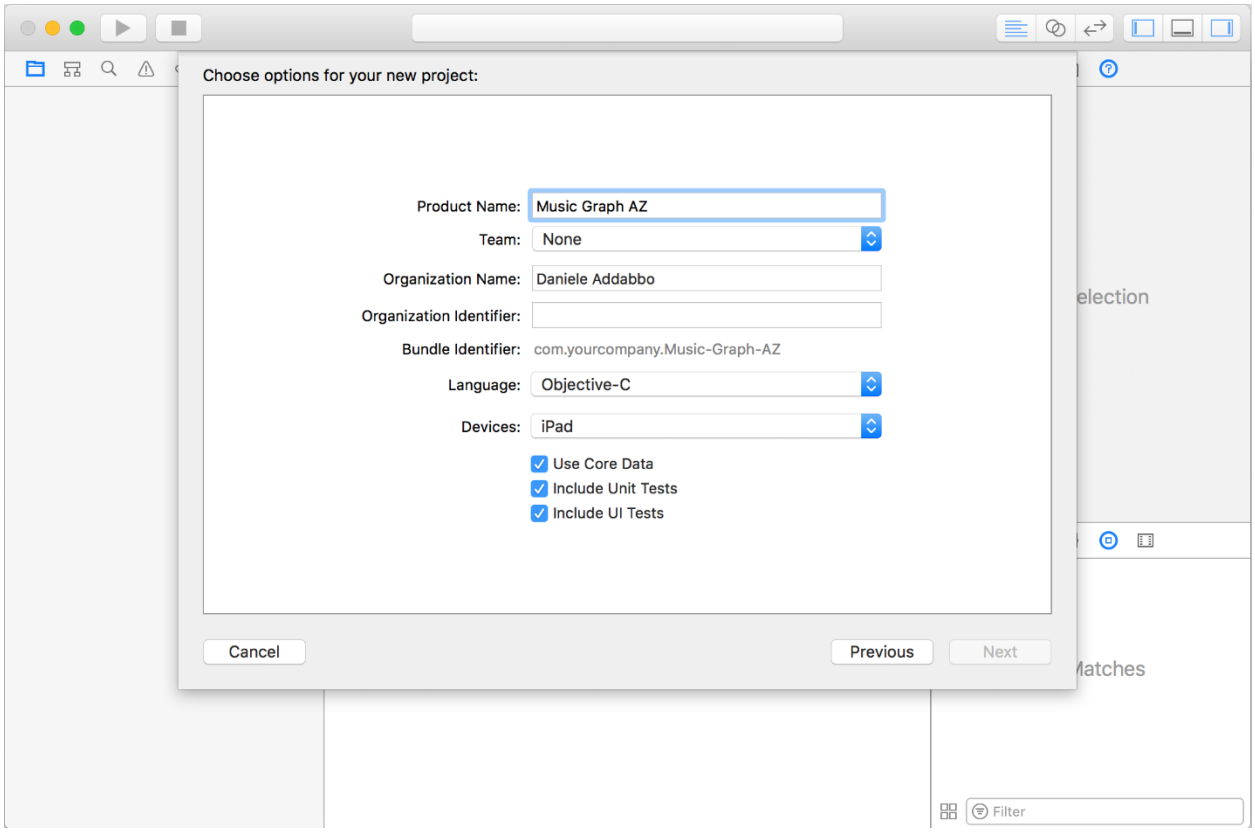


Fig. 16 Schermata di Xcode 8 per la scelta delle opzioni di sviluppo

Tra le altre opzioni che è necessario specificare durante la creazione di un progetto di Xcode è importante citare il Deployment Target. Con questo parametro si comunica ad Xcode quale versione di iOS deve essere supportata da questa applicazione. Per questo lavoro di tesi si è scelto di utilizzare come Deployment Target iOS 10.1.

5.3 Libreria file multimediali di iPad

Gli iPad sono device con una forte propensione alla multimedialità, per questo motivo possono ospitare contenuti multimediali di diverse tipologie. I brani musicali, gli AudioBook, i Podcast, i video musicali ed altri tipi di contenuti multimediali si trovano all'interno della Libreria file multimediali di iPad (Media Library nella versione inglese). Il sistema operativo iOS permette alle App di accedere alla libreria file multimediali.

5.3.1 Novità di iOS 10

L'ultima versione di iOS, la numero 10, presenta alcune importanti novità relative alla libreria file multimediali di iPad (user's media library).

La più importante novità riguarda la privacy. Fino ad iOS 9.3 (la versione precedente di iOS) tutte le App installate nell'iPad avevano libero accesso alla libreria file multimediali senza che l'utente ne fosse a conoscenza. Questa libertà di accesso alla libreria permetteva potenzialmente degli usi poco leciti delle informazioni personali dell'utente. Ad esempio una App anche se non pensata per riprodurre musica, poteva analizzare la libreria musicale per analizzare i gusti dell'utente. Un altro esempio di uso malevolo delle informazioni riguarda la possibilità per due App dello stesso produttore di analizzare tutta la libreria multimediale dell'utente per identificare l'utente stesso.

A partire da iOS 10.0 per accedere alla libreria file multimediali di iPad è necessario inserire una nuova voce nel file Info.plist del progetto. Questo file è un Information Property List File che è presente in tutti i progetto di Xcode. La nuova voce si chiama NSAppleMusicUsageDescription e consente di inserire una stringa di testo che verrà mostrata all'utente il quale sceglierà se consentire alla App di accedere alla libreria. Le App che hanno chiesto l'accesso alla libreria file multimediali vengono automaticamente inserite da iOS nella schermata Impostazioni Privacy del dispositivo. Tramite questa schermata l'utente può verificare e modificare l'elenco delle App con accesso alla libreria consentito o negato.

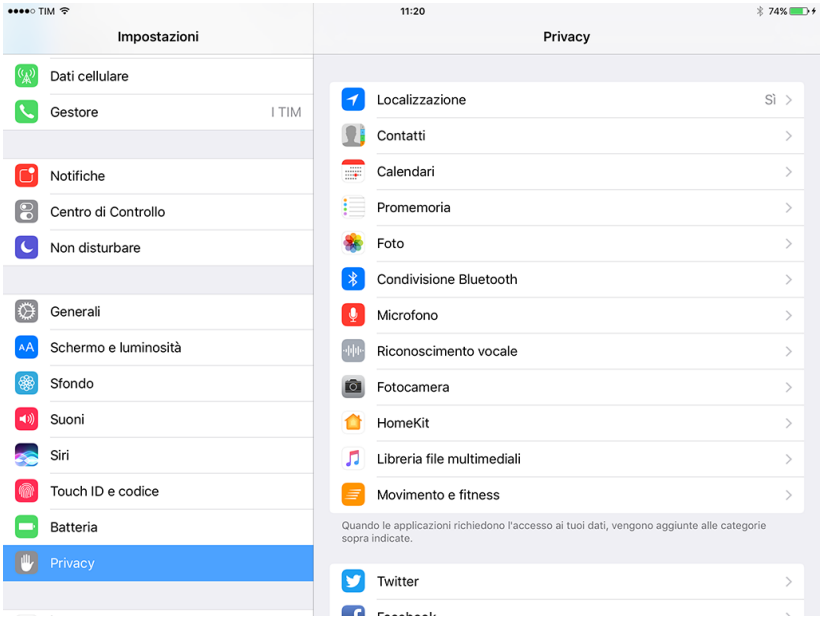


Fig. 17 Impostazioni Privacy di iOS 10

Se l'App non viene aggiornata per iOS 10, al primo accesso alla libreria file multimediali iOS termina la App. Lo sviluppatore può vedere questo messaggio di errore in Xcode:

This app has crashed because it attempted to access privacy-sensitive data without a usage description. The app's Info.plist must contain an NSAppleMusicUsageDescription key with a string value explaining to the user how the app uses this data.

Il codice Objective-C per verificare se l'App ha accesso alla libreria file multimediali di iOS è:

```
if ([MPMediaLibrary authorizationStatus]==MPMediaLibraryAuthorizationStatusAuthorized)  {
        // sì
} else {
        // no
}
```

Il codice per chiedere esplicitamente l'accesso alla libreria file multimediali di iOS è:

```
[MPMediaLibrary requestAuthorization:^(MPMediaLibraryAuthorizationStatus stato){
        switch(stato){
                case MPMediaLibraryAuthorizationStatusAuthorized:
                // sì
                break;
            case MPMediaLibraryAuthorizationStatusRestricted:
                // no
                break;
            case MPMediaLibraryAuthorizationStatusDenied:
                // no
                break;
            default:
                break;
        }
}];
```

5.3.2 I Framework per l'accesso alla Media Library

Apple mette a disposizione degli sviluppatori diversi Framework e diverse API (application programming interface) per l'accesso alla Media Library. I tre Framework più importanti sono: AV Foundation Framework, Media Player Framework e AV Kit Framework.

Come vedremo nel prossimo capitolo 5.4, l'elenco delle librerie è piuttosto vasto. Ma tra tutte, ce n'è una che si distingue per la possibilità di effettuare con semplicità ricerche all'interno della Media Library. Stiamo parlando di Media Player Framework. Questa è la tecnologia che è stata scelta per questo lavoro di tesi per effettuare le ricerche all'interno della Media Library.

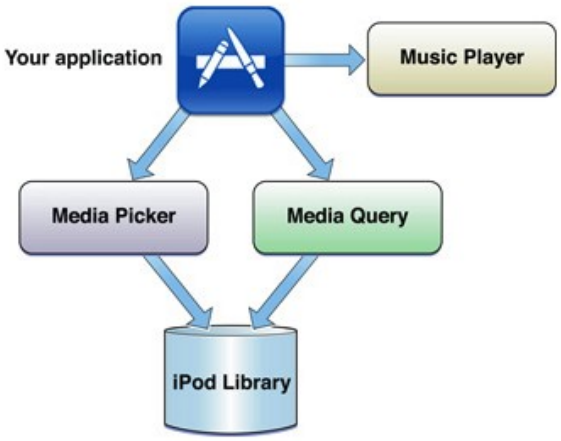


Fig. 18 Accedere alla Media Library

Questa immagine mostra le due metodologie di accesso alla Media Library offerte da Media Player Framework [24]: tramite il Media Picker oppure tramite una Media Query.

Il Media Picker è sostanzialmente l'interfaccia dell'applicazione Musica di Apple resa disponibile per consentire all'utente dell'applicazione che si sta sviluppando di selezionare uno o più brani. Questa soluzione ha il vantaggio di consentire all'utente di usare un'interfaccia familiare, ma non consente di effettuare ricerche avanzate.

La Media Query è un sistema di interrogazione che consente di usare la Media Library (o iPod Library) come un database. Ad esempio queste linee di codice mostrano l'elenco dei brani della libreria che contengono all'interno del nome un testo digitato dall'utente:

```
NSString *testo = self.canzoneTextField.text;
MPMediaQuery *everything=[MPMediaQuery songsQuery];

MPMediaPropertyPredicate *predicatoNome=[MPMediaPropertyPredicate predicateWithValue:testo
forProperty:MPMediaItemPropertyTitle comparisonType:MPMediaPredicateComparisonContains];
```

```
[everything addFilterPredicate:predicatoNome];

NSArray *itemsFromGenericQuery = [everything items];

for (MPMediaItem *song in itemsFromGenericQuery) {

        NSString *songTitle = [song valueForProperty: MPMediaItemPropertyTitle];
        NSLog (@"%@", songTitle);

}
```

In questo esempio viene usata la query predefinita songQuery che contiene l'elenco di tutti i brani della libreria. A questa query viene poi applicato un predicato che serve per filtrare la query in modo che restituisca solo i brani che contengono nel titolo il testo digitato dall'utente.

E' importante notare che nella iPod Library i brani sono rappresentati da oggetti della classe MPMediaItem. I brani non sono divisi per artista o per album, sono tutti salvati individualmente.

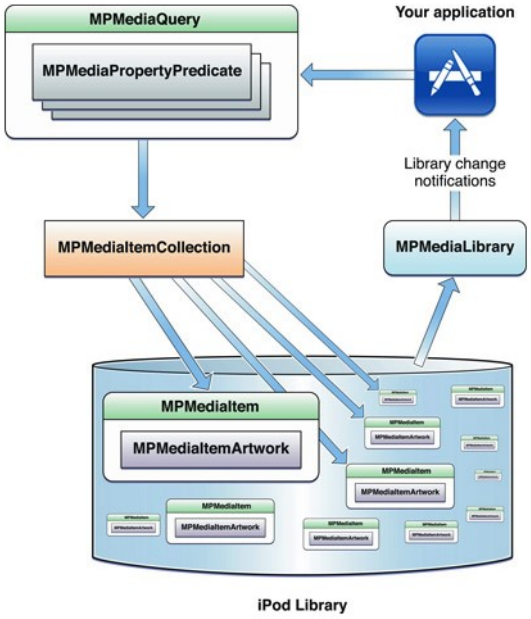


Fig. 19 Organizzazione della Media Library

La classe MPMediaItem contiene decine di proprietà che consentono di identificare il brano. Ad esempio il titolo (title), il nome dell'album (albumTitle), il nome dell'artista (artist). Tra queste proprietà ce ne sono due particolarmente importanti:

1) assetURL: rappresenta l'indirizzo del brano. Se si parla di brani salvati all'interno di un device (ad esempio un iPad) rappresenta il percorso per trovare un brano all'interno del file system.
2) persistentID: si tratta di un numero che identifica in modo univoco un brano. Sebbene non ci sia garanzia che questo ID non venga cambiato nel lungo periodo, è comunque un modo molto importante per identificare i brani, ad esempio per poterli cercare oppure per salvare

all'interno dell'App alcune informazioni da associare ad un brano. Questo ID è univoco all'interno della libreria ma lo stesso brano può avere ID diversi in device diversi.

A partire da iOS 5.0 un oggetto della classe MPMediaItem più rappresentare anche un video.

Media Player Framework è una libreria che esiste da molto tempo (la sua prima versione risale ad iOS 2.0). E' ancora oggi lo strumento più adatto effettuare ricerche nella Media Library ma non è più uno strumento consigliato per la riproduzione musicale o video. A partire da iOS 9 la riproduzione video tramite Media Player Framework è addirittura deprecata da Apple.

Per questo motivo si è scelto di usare per questo lavoro di tesi Media Player Framework per la ricerca ma non per la riproduzione audio.

5.4 Librerie audio

Ogni anno Apple presente nuove tecnologie relative ad iOS, solitamente durante il WWDC (Apple Worldwide Developers Conference) una conferenza destinata agli sviluppatori che si tiene ogni anno (solitamente a Giugno) in California.

Durante il WWDC vengono presentate novità ed evoluzioni relative alle API. In particolare per quanto riguarda l'audio, questa è la situazione delle librerie Apple aggiornata al WWDC 2106 [25]:

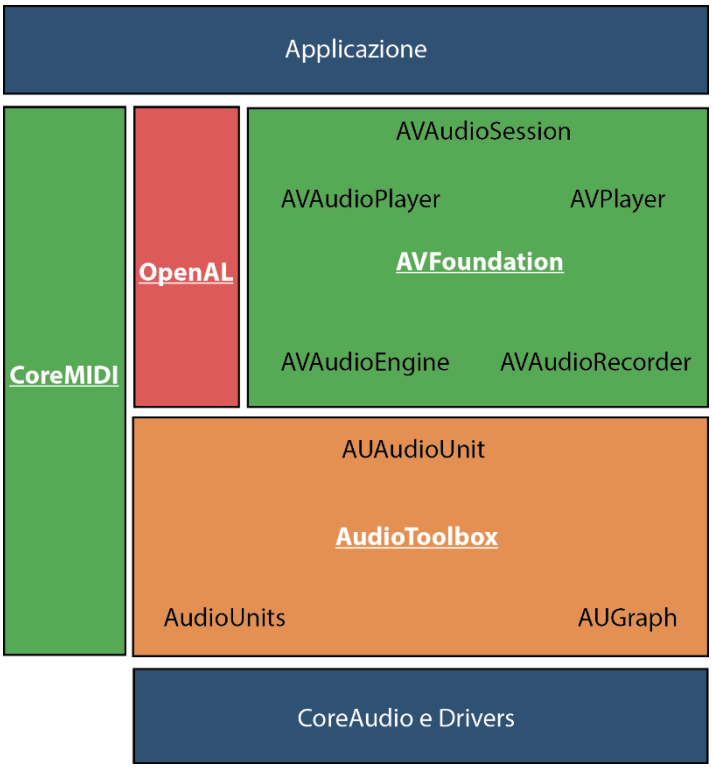


Fig. 20 Le principali librerie Audio in iOS 10

Questa figura presenta l'elenco delle librerie attualmente suggerite da Apple per la riproduzione musicale sui dispositivi iOS. La figura non include Media Player Framework in quanto considerato superato per la riproduzione audio e non include AVKit Framework in quanto più adatto alla riproduzione video (benché possa essere usato anche per la riproduzione di un brano musicale).

Tra le librerie rappresentate in figura, alcune sono sconsigliate in quanto superate da nuove librerie. Stiamo parlando di OpenAL e di AUGraph entrambe sostituite da AVAudioEngine [25].

AudioToolbox è un framework di basso livello che è consigliato soprattutto per applicazioni real time. Alcune delle funzioni che una volta erano prerogativa di AudioToolbox sono oggi disponibili in AVAudioEngine.

Core MIDI è un framework che serve per interfacciarsi con apparecchiature MIDI come ad esempio sintetizzatori.

AVFoundation è un framework che contiene diverse tecnologie al suo interno. AVAudioSession è una classe che serve a comunicare ad iOS che tipo di applicazione audio si sta sviluppando. AVPlayer è una classe adatta a riprodurre audio e video. AVAudioPlayer è una classe adatta a riprodurre solo audio [26]. AVAudioRecorder è una classe per la registrazione audio. AVAudioEngine è una tecnologia per la riproduzione audio più avanzata rispetto alle altre possibilità offerte da AVFoundation.

Per questo lavoro di tesi si è scelto di utilizzare AVAudioSession ed AVAudioEngine.

5.4.1 AVAudioSession

Gli utilizzatori di smartphone e tablet utilizzano questi device ovunque e nelle situazioni più diverse. Per questo motivo è opportuno che l'utente abbia un'esperienza consistente e prevedibile quando usa una App. Ad esempio se l'utente di uno smartphone sta ascoltando della musica e riceve una telefonata, si aspetta che la musica venga interrotta durante la telefonata e che poi riprenda automaticamente al termine della stessa. Per questo motivo esistono le Audio Session di iOS. Tramite AVAudioSession la App comunica ad iOS che tipo di utilizzo la App fa dell'audio.

Ad ogni applicazione che riproduce o registra audio in iOS viene automaticamente associata una Audio Session con dei valori di default. Questi valori di default, però, non sono adatti per un'applicazione come quella che è oggetto di questo lavoro di tesi. Ad esempio, i valori di default di Audio Session fanno in modo che non appena la App passa in background, cioè non è più visibile

sullo schermo, l'audio venga interrotto. Questo comportamento non è accettabile per un player musicale.

E' possibile personalizzare la Audio Session associata ad una App tramite la classe AVAudioSession.

Per impostare una AVAudioSession bisogna seguire cinque importanti passaggi:

1) **Ottenere un riferimento alla AVAudioSession della App.**

   Il codice è il seguente:

   ```
   AVAudioSession *session=[AVAudioSession sharedInstance];
   ```

2) **Impostare categoria, modo ed eventuali opzioni della AVAudioSession.**

   Esistono attualmente sette categorie tra le quali scegliere. La categoria di default è AVAudioSessionCategorySoloAmbient ma come abbiamo visto non è adatta ad un player musicale. La categoria pensata per un player musicale è AVAudioSessionCategoryPlayback.

   Quando si seleziona questa categoria il suono riprodotto dalla App continua anche quando il tasto del device per passare alla modalità silenziosa (tasto software nel caso di iPad) viene attivato. Questa categoria consente di riprodurre musica anche quando la App si trova in background ma per poterlo fare è necessario aggiungere anche una voce nel file Info.plist del progetto. La voce da aggiungere si chiama UIBackgroundModes. Di default questa categoria è di tipo nonmixable, questo significa che all'apertura di una App appartenente a questa categoria tutte le altre sessioni audio (di altre App) che sono non mixabili vengono interrotte. E' possibile cambiare questo comportamento con l'opzione AVAudioSessionCategoryOptionMixWithOthers. Vedere il punto 3 per ulteriori dettagli.

   Il codice usato per questo progetto è il seguente:

```
    NSError *setCategoryError = nil;
BOOL success = [session setCategory:AVAudioSessionCategoryPlayback error:&setCategoryError];
     if(!success)
     {
            NSLog(@"Errore: setCategory");
     }
  NSError *setModeError=nil;
  success = [session setMode:AVAudioSessionModeDefault error:&setModeError];
 if(!success)
 {
     NSLog(@"Errore: setMode");
 }
```

3) Attivare la sessione.

Senza questo comando le impostazioni che si sono scelte non vengono applicate. Una volta attivata la sessione, in base alle impostazioni che si sono scelte, la Audio Session determina l'interazione tra questa App e le altre App. Ad esempio, supponiamo che la App "Musica" (una delle App preinstallate da Apple) stia riproducendo un brano. A questo punto viene lanciata una App con categoria AVAudioSessionCategoryPlayback e con modo AVAudioSessionModeDefault che è di tipo nonmixable. Non appena questa App attiva la propria Audio Session questa operazione interrompe la riproduzione da parte della App "Musica" [25].

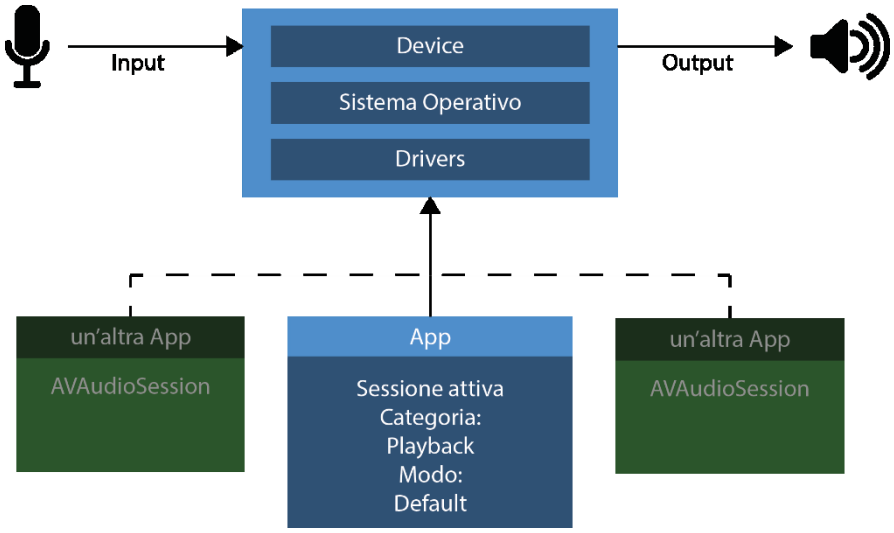


Fig. 21 AVAudioSession attiva

Il codice per attivare la sessione è il seguente:

```
NSError *setActiveError=nil;
  success=[session setActive:YES error:&setActiveError];
 if(!success)
  {
  NSLog(@"ERRORE: setActive");
 }
```

4) Gestire le interruzioni.

La sessione audio può essere interrotta, ad esempio da audio con priorità più alta: un tipico esempio è il suono di allarme del timer. L'interruzione rende la sessione inattiva ed interrompe l'audio ancora prima che la App riceva la notifica. Quando l'interruzione è finita bisogna gestire la situazione in modo appropriato, ad esempio riattivando la sessione audio e facendo ripartire l'audio [27].

Per gestire le interruzioni bisogna registrare la App per ricevere le notifiche.

Il codice è il seguente:

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(gestioneAudioSessionInterruption:) name:AVAudioSessionInterruptionNotification
object:session];
   [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(gestioneMediaServicesReset:)
name:AVAudioSessionMediaServicesWereResetNotification object:session];
```

Il codice che segue mostra come il metodo che si deve occupare della InterruptionNotification gestisce i vari casi di notifica:

```
-(void)gestioneAudioSessionInterruption:(NSNotification *)notification
{
   NSNumber *interruptionType=[[notification userInfo]
objectForKey:AVAudioSessionInterruptionTypeKey];
   NSNumber *interruptionOption=[[notification userInfo]
objectForKey:AVAudioSessionInterruptionOptionKey];

   switch(interruptionType.unsignedIntegerValue){
      case AVAudioSessionInterruptionTypeBegan:
         NSLog (@"inizio interruzione");
         // audio già fermato, già inattivo
         break;
      case AVAudioSessionInterruptionTypeEnded:
         NSLog (@"fine interruzione");
         // se si vuole far ripartire la musica sempre
         [self.playerTest play];
         // se si vuole far ripartire la musica solo in un caso specifico
         if(interruptionOption.unsignedIntegerValue==AVAudioSessionInterruptionOptionShouldResume){
            [self.playerTest play];
         }
         break;
   } // switch
}
```

E' importante notare che non tutte le interruzioni che hanno un inizio (TypeBegan) hanno anche una fine (TypeEnded).

5) Gestire i cambiamenti di periferiche audio (Route Changes).

Tutti gli smartphone e tablet ed in particolare quelli basati sul sistema operativo iOS seguono delle consuetudini nella gestione delle periferiche audio che ogni App dovrebbe rispettare.

Ad esempio, quando un utente che sta ascoltando della musica collega le cuffie al suo dispositivo, l'utente si aspetta che l'audio, che prima veniva riprodotto tramite gli altoparlanti del proprio device, venga ora riprodotto dalle cuffie e che la musica continui. Al contrario quando l'utente scollega le cuffie si aspetta che l'audio venga interrotto.

Per gestire questo tipo di variazioni, è necessario configurare la App per ricevere queste notifiche.

Il codice è il seguente:

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(gestioneRouteChange:)
name:AVAudioSessionRouteChangeNotification object:session];
```

Il codice per gestire i singoli casi di cambiamento di periferica audio è il seguente:

```
-(void)gestioneRouteChange:(NSNotification *)notification
{
   NSNumber *motivoRouteChange=[[notification userInfo]
valueForKey:AVAudioSessionRouteChangeReasonKey];

   switch(motivoRouteChange.unsignedIntegerValue){
      case AVAudioSessionRouteChangeReasonNewDeviceAvailable:
         NSLog (@"nuova periferica audio disponibile (ad esempio: cuffie)");
         break;
      case AVAudioSessionRouteChangeReasonOldDeviceUnavailable:
         NSLog(@"la vecchia periferica audio non è più disponibile");
         // se si vuole far ripartire la musica dopo la disconnessione
         // [self.playerTest play];
         break;
      case AVAudioSessionRouteChangeReasonCategoryChange:
         NSLog(@"category change: la categoria del session object è cambiata");
         break;
   }
}
```

5.4.2 AVAudioEngine

Nel 2014 Apple ha introdotto, con iOS 8, AVAudioEngine, una nuova API (application programming interface) all'interno del framework AV Foundation. Per la gestione dell'audio prima dell'introduzione di AVAudioEngine erano disponibili API di alto livello, con le quali era possibile solo eseguire semplici operazioni, oppure API di basso livello che consentivano una completa gestione dell'audio ma che erano difficili da gestire e richiedevano molte righe di codice.

AVAudioEngine, introdotta in iOS 8 e poi ulteriormente sviluppata nelle versioni successive di iOS, è nata con l'intenzione di fornire agli sviluppatori la possibilità di eseguire funzioni complesse in modo semplice. Tra le caratteristiche principali di questa API ricordiamo [28]:

- Basso tempo di latenza e gestione dell'audio in real time
- Riproduzione di audio da file oppure da buffer (per esempio da internet)
- Registrazione di audio
- Gestione del flusso audio in blocchi di elaborazione, chiamati nodi, collegabili tra loro
- Possibilità di registrare l'audio in qualsiasi punto della catena di elaborazione
- Possibilità di implementare audio 3D

L'elemento principale di questa API è la classe AVAudioEngine. Una istanza di questa classe rappresenta un contenitore che verrà poi utilizzato per collegare i nodi della catena di gestione del flusso audio. AVAudioEngine gestisce un grafo di nodi audio e consente una gestione dinamica del collegamento tra i nodi. I nodi, che sono sottoclassi di AVAudioNode, possono essere principalmente di tre tipi:

- Nodi di input, ad esempio appartenenti alla classe AVAudioPlayerNode
- Nodi di elaborazione, ad esempio appartenenti alla classe AVAudioMixerNode oppure alla classe AVAudioUnitEffect
- Nodi di uscita (output) appartenenti alla classe AVAudioOutputNode

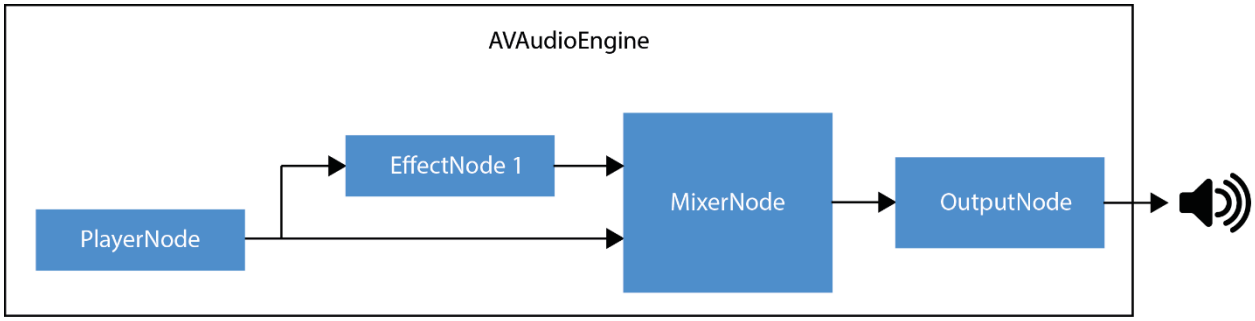


Fig. 22 Istanza di AVAudioEngine con alcuni nodi

Per utilizzare AVAudioEngine bisogna importare il framework AVFoundation con il codice:

```
#import <AVFoundation/AVFoundation.h>
```

Bisogna poi seguire cinque passaggi:

1) **Creare un motore AVAudioEngine.**

   Quando si crea un'istanza della classe AVAudioEngine si crea il contenitore per i nodi di elaborazione del flusso audio. Ogni istanza di AVAudioEngine ha due nodi automaticamente creati dalla classe: un nodo di output, appartenente alla classe AVAudioOutputNode, ed un nodo Mixer, appartenente alla classe AVAudioMixerNode. Questi due nodi sono automaticamente collegati tra loro. Non è possibile aggiungere altri nodi di output, ma è possibile aggiungere altri nodi mixer all'interno dello stesso motore (engine).

   Il codice per creare l'istanza di AVAudioEngine ed avere i riferimenti al nodo mixer creato automaticamente è il seguente:

```
AVAudioEngine *engine = [[AVAudioEngine alloc] init];
AVAudioMixerNode *mainMixer = [engine mainMixerNode];
```

2) Creare i nodi da utilizzare.

Il passo successivo è creare i nodi di input. Per esempio un nodo di tipo AVAudioPlayerNode per leggere i file audio della Media Library.

Il codice per creare il player è il seguente:

```
AVAudioPlayerNode *player1 = [[AVAudioPlayerNode alloc] init];
```

Per rendere il player più interessante e flessibile, si è deciso di sfruttare una nuova funzionalità di AVAudioEngine introdotta a partire da iOS 9 (2015) [29], che consente di collegare l'output di un nodo a più nodi. In questo caso si è scelto di collegare l'output del player direttamente al mixer ma anche ad un effetto.

Il codice per creare l'effetto, in questo esempio un filtro passa alto ed un equalizzatore, è il seguente:

```
AVAudioUnitEQ *eq = [[AVAudioUnitEQ alloc] initWithNumberOfBands:2];

AVAudioUnitEQFilterParameters *filterParameters = eq.bands[0];
filterParameters.filterType = AVAudioUnitEQFilterTypeHighPass;
filterParameters.frequency = 80;
filterParameters.bypass=false;

filterParameters = eq.bands[1];
filterParameters.filterType = AVAudioUnitEQFilterTypeParametric;
filterParameters.frequency = 500;
filterParameters.bandwidth = 2.0; // da 0.05 a 5.0 ottave
filterParameters.gain = 10.0; // da -96 a +24 db
filterParameters.bypass=false;
```

3) Inserire i nodi nel motore.

Dopo aver creato i nodi è fondamentale aggiungerli all'engine creato.

Il codice per aggiungere il player e l'effetto all'engine è il seguente:

```
[engine attachNode:player1];
[engine attachNode:eq];
```

4) Collegare i nodi tra loro.

Non è possibile utilizzare AVAudioEngine senza aver creato i collegamenti. In questo caso è necessario creare due collegamenti: il primo tra il player e l'effetto, il secondo tra il player ed il mixer. Il mixer di default (mainMixer) e l'output sono già collegati in automatico.

E' importante notare che quando l'audio di un nodo viene diviso tra due o più destinazioni, l'audio che viaggia verso le destinazioni è identico, non viene suddiviso in canali oppure diminuito di volume. Per collegare l'output del player a più nodi si usa la classe AVAudioConnectionPoint.

Il codice per collegare l'effetto al mixer è il seguente:

```
[engine connect:eq to:mainMixer format:file.processingFormat];
```

Il codice per collegare il player all'effetto ed anche al mixer è il seguente:

```
NSArray<AVAudioConnectionPoint *> *connPoints=[NSArray
arrayWithObjects:[[AVAudioConnectionPoint alloc] initWithNode:eq bus:0],[[AVAudioConnectionPoint
alloc] initWithNode:mainMixer bus:1], nil];

    [engine connect:player1 toConnectionPoints:connPoints fromBus:0 format:file.processingFormat];
```

5) **Far partire il motore.**

   L'ultima operazione necessaria è far partire il motore dell'istanza di AVAudioEngine. Il codice che segue mostra come creare un riferimento ad un file (usando un percorso di file "anUrl" precedentemente ottenuto), come caricare il file nel player, come avviare l'engine e come far partire la riproduzione:

```
AVAudioFile *file = [[AVAudioFile alloc] initForReading:anUrl error:&error];
[player1 scheduleFile:file atTime:nil completionHandler:nil];
[engine startAndReturnError:&error];
[player1 play];
```

Le possibilità offerte da AVAudioEngin sono notevoli. E' possibile aggiungere senza limitazioni all'engine il numero di nodi che si desidera usare. Ad esempio è possibile usare più player, per creare ad esempio un effetto dissolvenza con sovrapposizione delle tracce. Oppure si può modificare il suono tramite effetti. Inoltre è possibile registrare il suono su file in qualsiasi punto del flusso audio.

5.5 Grafi per la rappresentazione della libreria musicale

Per rappresentare la libreria musicale sotto forma di grafo è necessario compiere delle scelte importanti per quanto riguarda il tipo di informazione che si vuole sia rappresentato, questo anche per evitare che il numero eccessivo di nodi crei un grafo difficile da leggere e da visualizzare.

E' necessario poi compiere delle scelte relativamente al modo di collegare i nodi tra loro. Questo significa stabilire quali sono le fonti che stabiliscono le relazioni tra i nodi.

E' poi necessario scegliere se la rappresentazione su schermo della libreria musicale deve essere costituita da un solo grande grafo oppure da più grafi.

5.5.1 Tipologia dei nodi

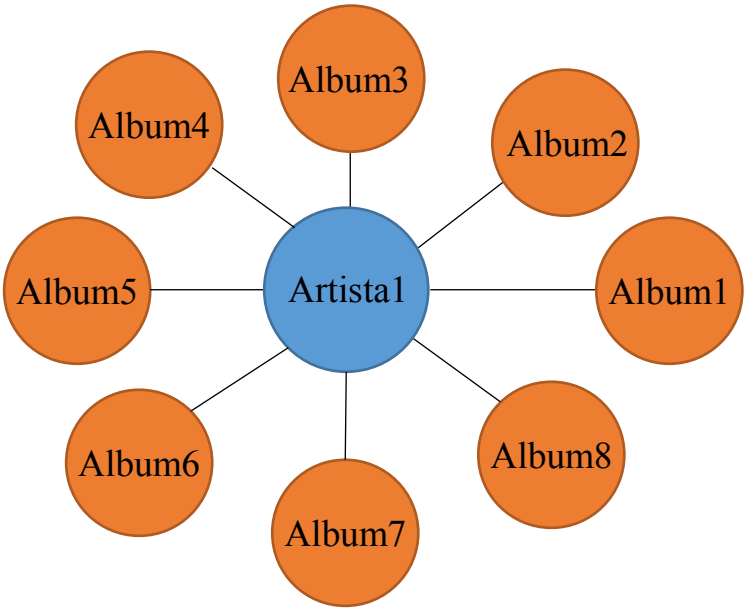
Per la rappresentazione grafica della libreria musicale in questo lavoro di tesi si è scelto di utilizzare i grafi.

Gli elementi della libreria che si è scelto di rappresentare graficamente sono:

- Artisti
- Album
- Brani musicali o canzoni

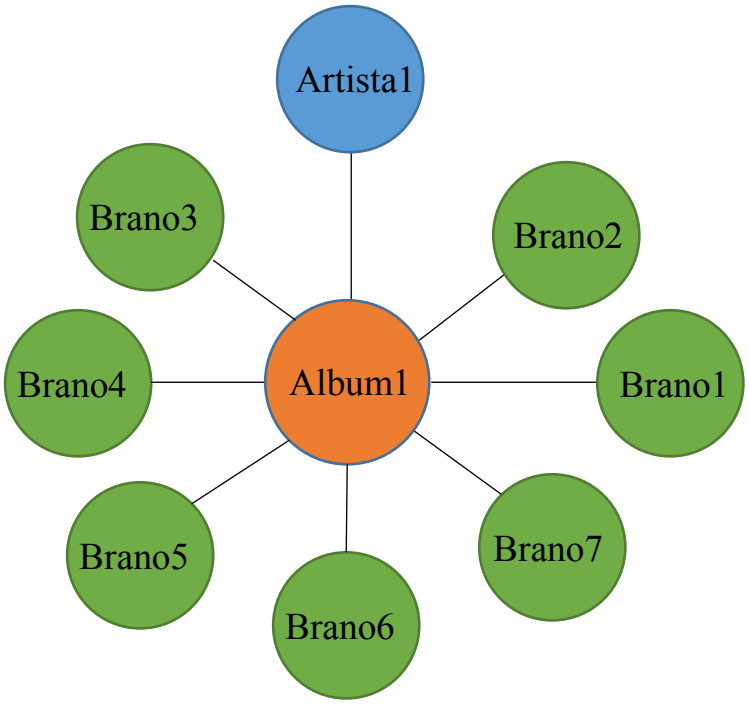
Quando un utente inizia ad usare la App oggetto di questo lavoro di tesi, ha la possibilità di scegliere da quale nodo vuole iniziare a rappresentare il grafo. A seconda della scelta compiuta dall'utente il risultato della rappresentazione è differente.

Se l'utente scegliere di partire da un artista la rappresentazione grafica sarà la seguente:



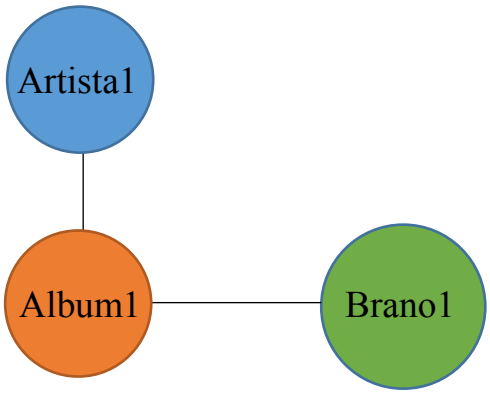
Intorno al nome dell'artista vengono mostrati i nodi corrispondenti agli album dell'artista.

Se l'utente scegliere di partire da un album, la rappresentazione grafica sarà la seguente:



Se disponibile, la copertina dell'album viene mostrata insieme al nome. Intorno al nome dell'album vengono mostrati i nodi corrispondenti ai brani dell'album.

Se l'utente sceglie di partire da un brano, la rappresentazione grafica sarà la seguente:



Intorno al nome del brano vengono mostrati i nodi corrispondenti all'album che contiene il brano e all'artista.

Ulteriori collegamenti tra nodi sono disponibili nell'App. Verranno analizzati nei prossimi paragrafi.

5.5.2 Collegamenti base tra i nodi

L'uso dei metadati della libreria musicale dell'iPad consente di creare collegamenti tra artisti, album, brani musicali e compositori. I collegamenti tra questi elementi è consentito dalla presenza all'interno della libraria musicale, di numeri identificativi (ID) che permettono di identificare univocamente gli elementi della libreria. Ad ogni brano, se disponibili, possono essere associati i seguenti ID:

Proprietà	Significato
MPMediaItemPropertyPersistentID	ID che identifica il brano
MPMediaItemPropertyAlbumArtistPersistentID	ID che identifica l'artista principale dell'album
MPMediaItemPropertyArtistPersistentID	ID che identifica l'artista del brano
MPMediaItemPropertyAlbumPersistentID	ID che identifica l'album
MPMediaItemPropertyGenrePersistentID	ID che identifica il genere musicale
MPMediaItemPropertyPodcastPersistentID	ID che identifica un podcast
MPMediaItemPropertyComposerPersistentID	ID che identifica il compositore

Questi attributi fanno parte dei metadati associati al brano.

Esistono però alcune limitazioni da prendere in considerazione. Ad esempio l'ID che identifica il compositore non è sempre presente, per cui va usato solo se disponibile. Inoltre questi ID sono univoci all'interno della libreria sul singolo device ma non sono univoci se si confrontano librerie di due differenti device (iPad). Questo limita molto le possibilità di utilizzo di queste informazioni.

Oltre ai collegamenti tra nodi consentiti dagli ID della libreria, si è scelto di creare collegamenti tra nodi in altri due modi:

- Analisi dei metadati
- Metadati reperibili da Internet

5.5.3 Collegamenti avanzati tra i nodi basati su metadati

Dall'analisi approfondita dei metadati è possibile generare altri lati di collegamento tra nodi. Ad esempio facendo una ricerca in base al nome del brano e al compositore è possibile verificare se all'interno della libreria ci sono altre versioni del brano magari appartenenti ad artisti diversi. Oppure facendo una ricerca in base al nome del brano, all'artista e alla durata è possibile verificare se all'interno della libreria lo stesso brano è contenuto in più album.

La App offre due tipi di collegamenti avanzati tra nodi basati su metadati:

1) **Cover**

Ogni volta che il nodo di un brano viene mostrato sullo schermo, la App verifica, tramite un apposito metodo, se nella libreria musicale dell'iPad sono presenti una o più cover dello spesso brano. Per cover si intende la reinterpretazione di brani musicali da parte di qualcuno che non ne è l'interprete originale.

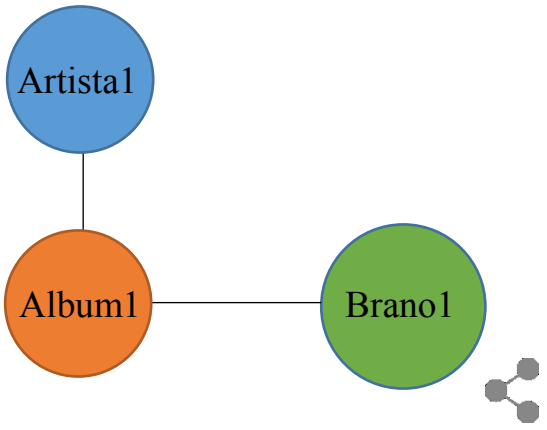
Per la ricerca delle cover vengono utilizzati i seguenti metadati del brano:

Proprietà	Significato
MPMediaItemPropertyTitle	Il titolo del brano
MPMediaItemPropertyComposerPersistentID	ID che identifica l'autore del brano

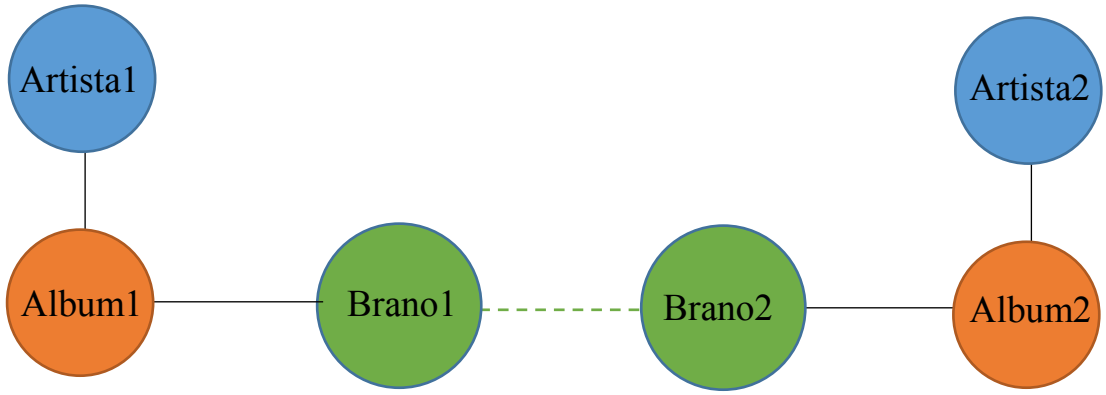
Se questa ricerca, applicata a tutti i brani della libreria, risulta positiva, i brani vengono memorizzati in un apposito vettore chiamato "nodiCollegati" associato al brano.

A livello grafico, la App mostra vicino al nodo del brano un apposito simbolo grafico che segnala che quel brano ha altri nodiCollegati.

Ecco un esempio di rappresentazione grafica di un brano con altri nodi da mostrare:



Dopo aver cliccato sul nodo del brano, vengono mostrati i brani cover collegati. Il collegamento tra i brani cover avviene tramite una linea tratteggiata di colore verde (come i nodi dei brani):



2) Brani identici

Ogni volta che il nodo di un brano viene mostrato sullo schermo, la App verifica, tramite un apposito metodo, se nella libreria musicale dell'iPad sono presenti uno o più brani identici a quello in esame. La presenza di brani identici nella libreria può essere, ad esempio, dovuta alla presenza di album originali e di raccolte dello stesso artista.

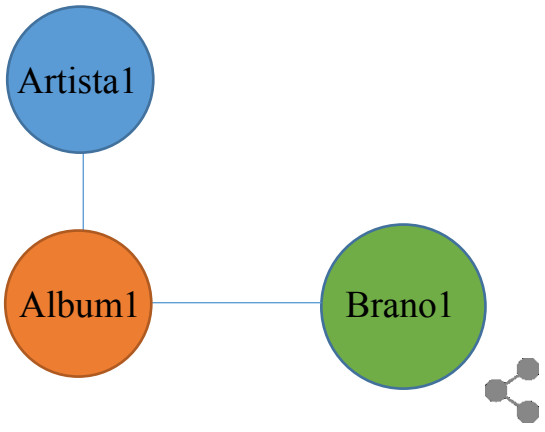
Per la ricerca dei brani identici vengono utilizzati i seguenti metadati del brano:

Proprietà	Significato
MPMediaItemPropertyTitle	Il titolo del brano
MPMediaItemPropertyArtistPersistentID	ID che identifica l'autore del brano
MPMediaItemPropertyPlaybackDuration	Durata del brano in secondi (float)

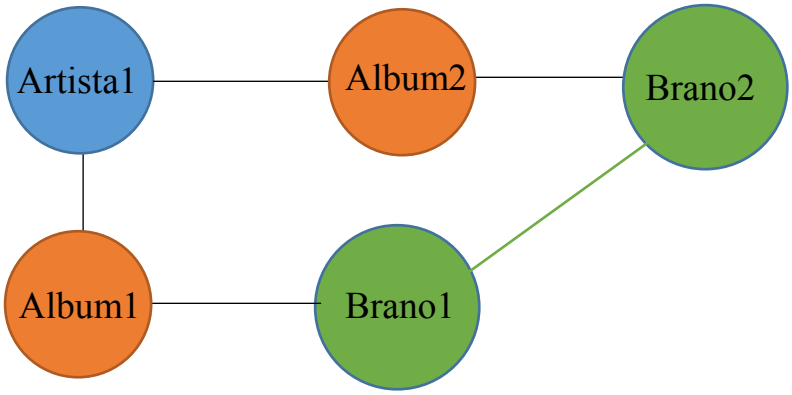
Se questa ricerca, applicata a tutti i brani della libreria, risulta positiva usando i primi due metadati come filtro, i brani vengono ulteriormente analizzati in base alla durata. Infatti i brani identici difficilmente hanno la stessa durata (espressa in millesimi di secondo). Quando un brano viene inserito all'interno di album differenti la sua durata può variare anche di alcuni secondi. Per questo motivo alla durata viene applicata una tolleranza di ±4.0 secondi. I brani identici vengono memorizzati in un apposito vettore chiamato "nodiCollegati" associato al brano.

A livello grafico, la App mostra vicino al nodo del brano un apposito simbolo grafico che segnala che quel brano ha altri nodiCollegati.

Ecco un esempio di rappresentazione grafica di un brano con altri nodi da mostrare:



Dopo aver cliccato sul nodo del brano, vengono mostrati i brani identici collegati. Il collegamento tra i brani identici avviene tramite una linea continua di colore verde (come i nodi dei brani):



5.5.4 Collegamenti avanzati tra i nodi con metadati da Internet

Esistono diverse banche dati musicali disponibili online, quali ad esempio Gracenote [30], AllMusic [31] e lo stesso iTunes Store della Apple. Si tratta soprattutto di banche dati o pensate per consentire agli utenti l'acquisto di musica online o comunque con accesso a pagamento per la App di terze parti.

Tra queste banche dati è importante segnalare la banca dati dell'iTunes Store di Apple che è accessibile tramite una API. Si tratta di una banca dati che consente di ricevere informazioni relative ai brani e agli artisti. La banca dati è nazionale, quindi restituisce valori diversi a seconda della nazione specificata nella ricerca.

Dopo diversi confronti tra banche dati musicali disponibili online si è scelto di utilizzare per questo lavoro di tesi la banca dati online Discogs [32]. Si tratta di una banca dati nata nel 2000 ed in continuo aggiornamento ed espansione. L'accesso a questa banca dati è gratuito ed è consentito fare fino a 240 interrogazioni al minuto per indirizzo IP [33].

Il tema della banche dati online è molto vasto ma limitato dalla difficoltà di confrontare le informazioni contenute in diverse banche dati, dato che i numeri identificativi non corrispondono.

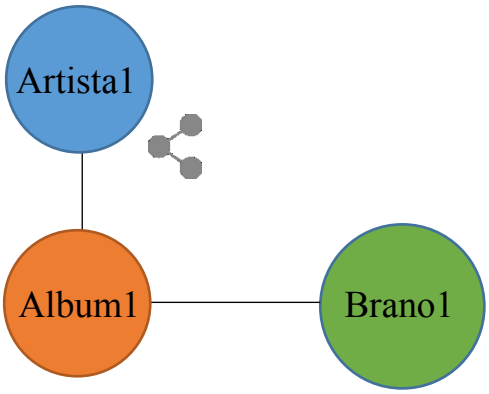
Si è scelto di aprire la App ai metadati reperibili da Internet, a titolo dimostrativo, per mettere in relazione gli artisti tra loro, collegando i gruppi musicali presenti nella libreria dell'iPad con i membri del gruppo, anch'essi presenti nella libreria dell'iPad, che hanno fatto anche una carriera come solisti.

Ogni volta che il nodo di un artista viene mostrato sullo schermo, la App verifica, tramite un apposito metodo, se nella libreria online Discogs esiste un artista con lo stesso nome. Le API di Discogs rispondono usando il formato JSON. Un volta trovato l'artista, la App esegue una seconda interrogazione a Discogs per aprire la pagina specifica dell'artista. Dentro a questa pagina, anch'essa ricevuta dalla App in formato JSON, viene effettuata la ricerca per capire se l'artista ha dei collegamenti con altri artisti.

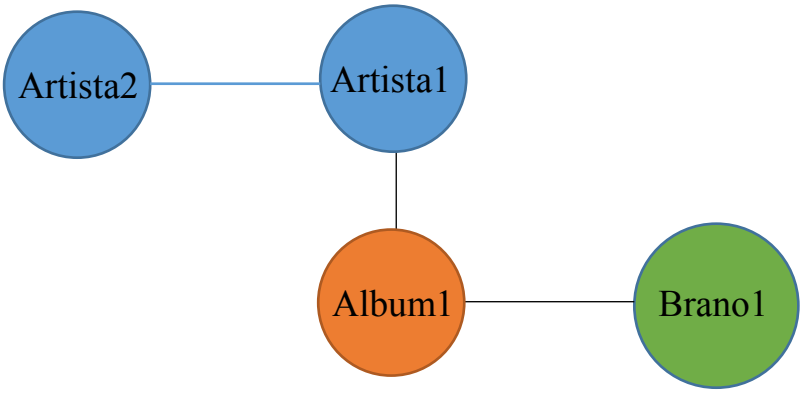
Per la ricerca di collegamenti tra artisti vengono utilizzati i seguenti metadati di Discogs:

Proprietà	Significato
title	Il nome dell'artista
members	Elenco di membri di un gruppo. Questa proprietà è vuota se si tratta di un artista
groups	Elenco dei gruppi nei quali l'artista ha lavorato

Ecco un esempio di rappresentazione grafica di un artista con altri nodi da mostrare:

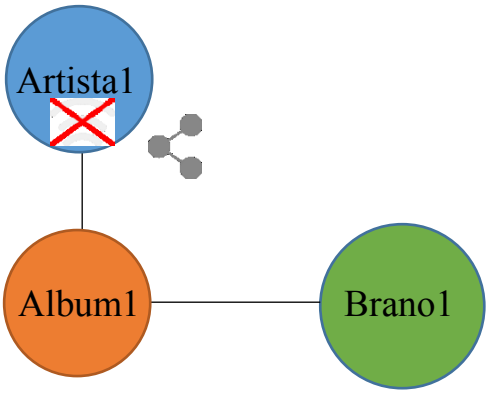


Dopo aver cliccato sul nodo dell'artista, vengono mostrati gli artisti collegati. Il collegamento tra artisti avviene tramite una linea continua di colore blu (come i nodi degli artisti):



Nel caso in cui, invece, il collegamento ad Internet non è disponibile, la App mostra un apposito simbolo sul nodo dell'artista ad indicare che durante l'ultimo tentativo di reperire informazioni su quell'artista c'è stato un problema. La App proverà ad effettuare un nuovo collegamento ad internet ogni volta che viene toccato il nodo dell'artista.

Ecco un esempio di rappresentazione grafica di un artista con altri nodi da mostrare ma con problemi di connessione ad internet:



5.5.5 Grafo connesso e non connesso

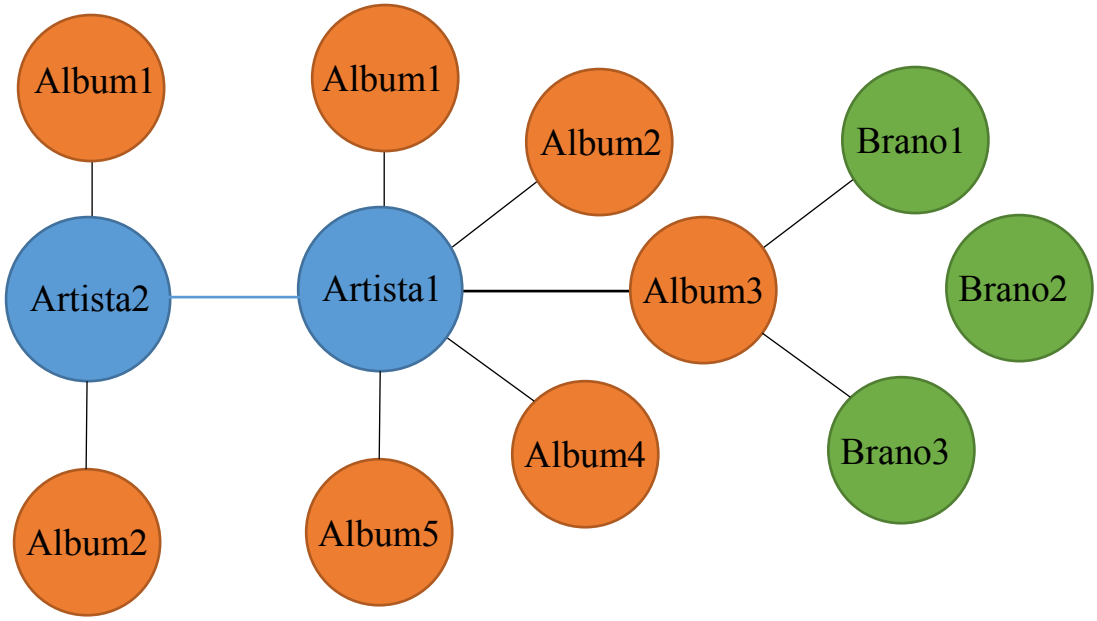
La rappresentazione dell'intera libreria musicale dell'iPad con un unico grafo all'apertura della App sarebbe possibile ma creerebbe un grafo di notevoli dimensioni e soprattutto poco pratico da utilizzare.

Per questo motivo si è scelto un approccio diverso, che parte da un semplice grafo per poi ampliare la navigazione in base alle scelte dell'utente.

Per aumentare le possibilità consentite all'utente si è scelto di consentire di partire da un grafo connesso per poi creare un grafo non connesso se l'utente inserisce nodi non collegati tra loro. In questo modo l'utente ha la possibilità di navigare la libreria in direzioni differenti, sempre mettendo in relazione i nodi dei differenti grafi tra loro, quando possibile.

Questo potrebbe essere un esempio di utilizzo della App: l'utente inizia ad usare la App creando un grafo dell'artista1 (ad esempio un gruppo), seleziona l'album3 in modo che i nodi corrispondenti ai brani dall'album vengano mostrati; a questo punto l'utente inserisce l'artista2 il quale ha i propri album ma ha anche collaborato con l'artista1. La App mostrerà l'apposito simbolo che segnala una possibilità di collegamento. Un volta cliccato sul nodo dell'artista si crea un collegamento tra l'artista2 e l'artista1 ed il grafo non connesso diventa un grafo connesso.

La rappresentazione grafica su schermo di quanto appena descritto potrebbe essere la seguente:



Quindi, pur avendo inserito nel grafo due artisti potenzialmente non collegati, dopo aver aperto il collegamento (artistico) tra questi artisti si ottiene un grafo connesso.

5.6 Librerie grafiche

Il sistema operativo iOS di Apple mette a disposizione diverse librerie grafiche. Ognuna di queste librerie grafiche ha una funzione specifica. Di seguito riportiamo un elenco di alcune delle più importanti librerie e framework grafici di iOS:

Nome	Da iOS	Descrizione
UIKit Framework	2.0	Framework usato per costruire e gestire l'interfaccia grafica della App (viste, pulsanti, …)
Core Graphics Framework	2.0	Framework di basso livello usato per grafica 2D
Core Animation	2.0	Infrastruttura per le animazioni.
SpriteKit	7.0	Framework di alto livello per la grafica 2D
SceneKit	8.0	Framework di alto livello per la grafica 3D

Per la realizzazione di questo lavoro di tesi era necessario trovare una libreria grafica che fosse in grado di rappresentare i grafi. Nessuna delle librerie fornite da Apple ha al suo interno i metodi necessari per la costruzione di un grafo. Non è stato inoltre possibile trovare una libreria grafica di terze parti in grado di rappresentare il grafo della libreria musicale.

Lo sviluppo di questa App ha richiesto quindi la realizzazione di un codice appositamente scritto per la rappresentazione dei grafi, dato che allo stato attuale non esistono librerie disponibili con tali funzionalità.

Dovendo comunque partire da librerie esistenti, si è scelto di basare il progetto su due librerie grafiche della Apple:

- UIKit per quello che riguarda l'interfaccia utente come pulsanti e menù;
- SpriteKit per tutto quello che riguarda la rappresentazione dei grafi.

In particolare ci soffermeremo brevemente nella descrizione della libreria SpriteKit.

5.6.1 SpriteKit

SpriteKit è un Framework introdotto da Apple con iOS 7 (2013). Si tratta di un framework pensato per l'animazione di immagini texture, dette anche sprite. SpriteKit consente inoltre di disegnare geometrie vettoriali. SpriteKit si occupa di ottimizzare il codice grafico in modo che le istruzioni di alto livello vengano poi convertite in codice di basso livello, tipicamente OpenGL o, più recentemente, Metal.

Il framework si basa sulla classe SKView alla quale si possono collegare una o più SKScene (si pensi ad esempio ai livelli di un videogioco). All'interno della SKScene la grafica viene costituita usando un vasto gruppo di classi. Nella seguente illustrazione abbiamo riportato le più importanti [34].

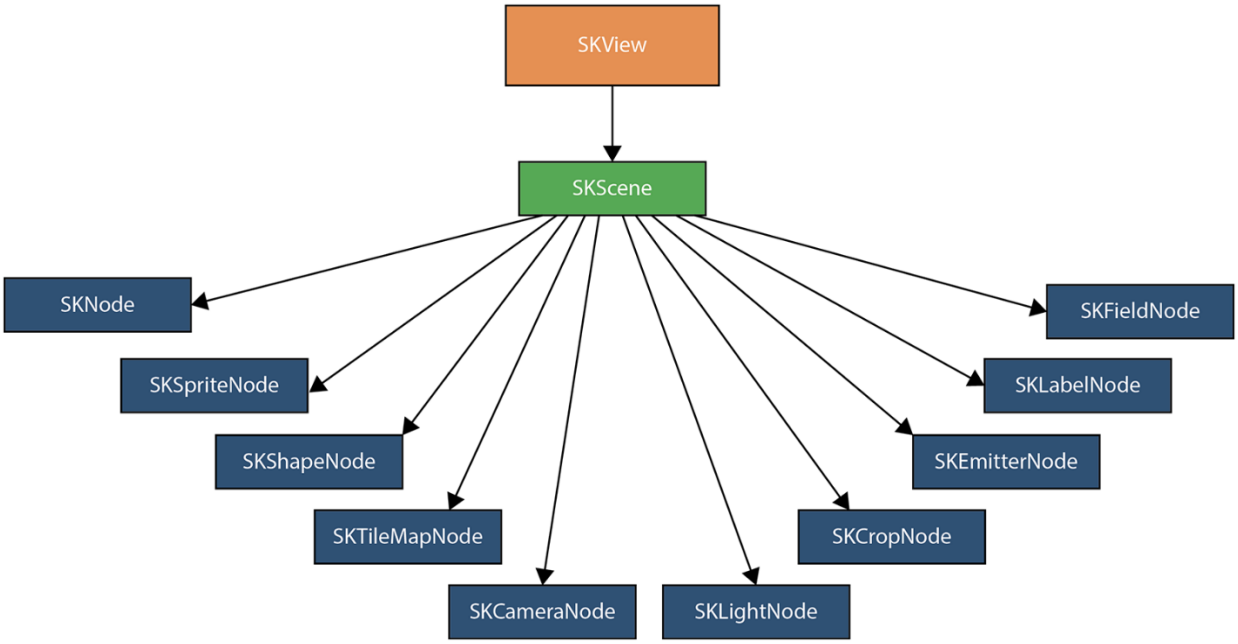


Fig. 23 SpriteKit Framework - classi principali

Non tutte queste classi sono state utilizzate in questo lavoro di tesi. Nel capitolo dedicato all'implementazione dei grafi entreremo nel dettaglio di quali classi sono state utilizzate.

Il framework SpriteKit è un framework pensato per la grafica 2D ed i videogiochi. Per questo motivo, pur essendo un framework di alto livello ha prestazioni molto elevate. Di default SpriteKit funziona ad un frame rate di 60 fps, questo significa che la scena (SKScene) viene aggiornata 60 volte ogni secondo.

SpriteKit, introdotto nel 2013 con iOS 7, si è evoluto molto durante gli ultimi anni. Ad esempio in iOS 9 (2015) è stata introdotta la classe SKCameraNode che consente di muovere una telecamera virtuale sulla scena, con la possibilità di regolare il livello di zoom e di rotazione dello schermo, questo consente di mostrare una porzione della scena oppure tutta la scena a discrezione del programmatore. Inoltre in iOS10 (2016) è stata introdotta la possibilità di impostare una proprietà di SKView chiamata preferredFramesPerSecond per ridurre il numero di fps, e allo stesso tempo, anche senza modificare i fps, SpriteKit riduce il proprio consumo di CPU di batteria in base alle reali necessità della scena.

Ogni volta che un frame deve essere calcolato e poi presentato su schermo, SpriteKit esegue una serie di metodi, rappresentati nella seguente figura, che il programmatore può decidere di usare per aggiornare il contenuto della scena sullo schermo. Tutto questo viene eseguito, con le impostazioni standard, 60 volte ogni secondo [35]:

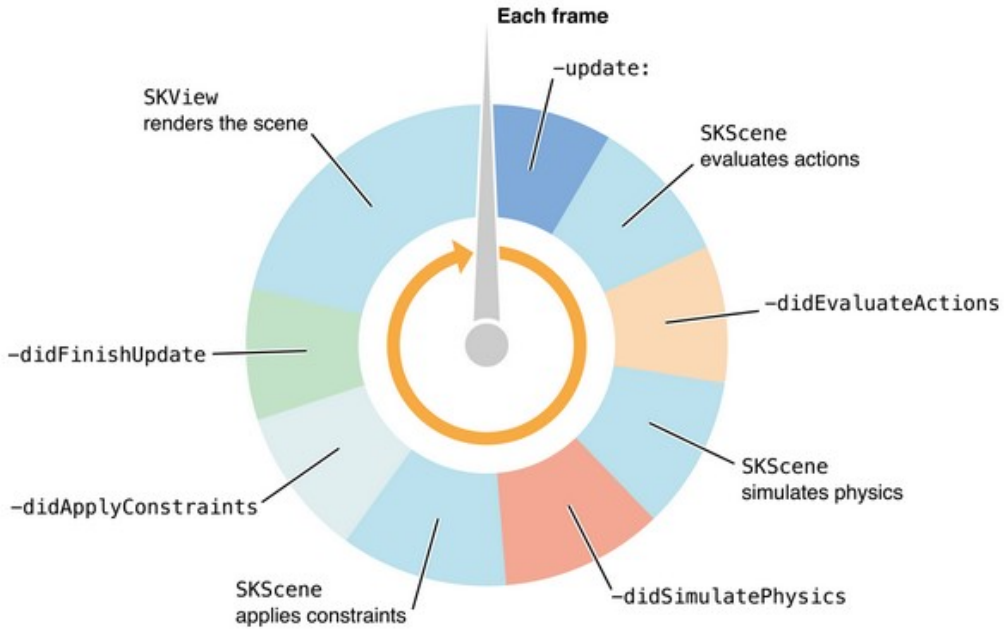


Fig. 24 Ciclo di refresh di un frame in SpriteKit

Quando si inserisce nella scena un oggetto rappresentato da una immagine, tipicamente si usa la classe SKSpriteNode, sottoclasse di SKNode. Ogni SKNode ha un frame, ovvero un rettangolo che contiene il nodo. Quando si lavora con le texture è molto importante comprendere il concetto di anchor point, vale a dire del punto del frame utilizzato per posizionare l'oggetto.

Di default il frame dello sprite e la sua texture sono centrati nella posizione dello sprite. E' comunque possibile fare in modo che una parte differente della texture compaia nella posizione del nodo. La proprietà anchorPoint determina quale punto del frame è posizionato nella posizione dello sprite.

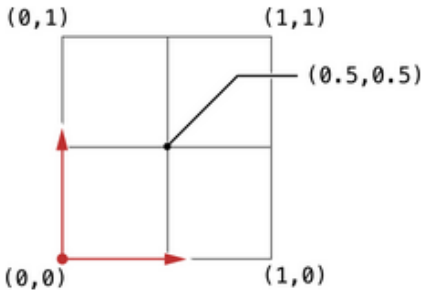


Fig. 25 SpriteKit - posizionamento del anchorPoint

Gli anchor point vengono specificati con un sistema di coordinate che va da 0 a 1 (vedi figura). L'origine si trova nell'angolo in basso a sinistra del frame e la posizione massima (1,1) si trova nell'angolo in alto a destra. Di default il valore di anchor point è (0.5, 0.5) che corrisponde al centro del frame.

Abbiamo visto che la scena (classe SKScene) viene usata per fornire il contenuto che deve essere mostrato dall'oggetto SKView. All'interno della scena, il contenuto viene creato come un albero di oggetti (nodi) la cui radice è la scena.

Per creare l'albero si utilizza la relaziona padre-figlio tra i nodi. Ogni nodo mantiene un lista ordinata dei figli, questa lista può essere letta attraverso la proprietà "children" del nodo. L'ordine dei figli all'interno dell'albero è importante e sia per l'elaborazione delle scena sia per la sua visualizzazione. Quindi l'ordine dei nodi va gestito con attenzione.

La tabella seguente mostra i più comuni metodi usati per costruire l'albero della scena.

Metodo	Descrizione
addChild:	Inserisce il nuovo nodo alla fine della lista dei figli del nodo padre
insertChild:atIndex:	Inserisce il nuovo nodo in una specifica posizione della lista dei figli del nodo padre
removeFromParent	Rimuove il nodo dalla lista dei figli del nodo padre

La seguente tabella mostra un elenco di proprietà utili per poter attraversare l'albero.

Proprietà	Descrizione
children	Il vettore degli oggetti SKNode che sono i figli di questo nodo
parent	Se questo nodo è figlio di un altro nodo, allora questa proprietà punta al padre. Diversamente il valore è nil
scene	Se il nodo è stato inserito in una scena, questa proprietà punta alla scena che è la radice dell'albero. Diversamente il valore è nil

Da quanto fino a qui scritto in relazione allo SpriteKit Framework, dovrebbe essere chiaro che si tratta di una libreria estremamente flessibile e di ottime prestazioni. Purtroppo questa libreria, come anche le altre fornite da Apple, non include direttamente le funzionalità necessarie per la rappresentazione dei grafi. Nonostante questo, considerando tutti i lati positivi di questo

framework, si è scelto di scrivere una nuova libreria specifica per la rappresentazione dei grafi basandosi sulla libreria SpriteKit.

Dato che SpriteKit non dispone di una struttura dati dedicata alla memorizzazione dei grafi, si è deciso di affiancare a SpriteKit un altro framework: GameplayKit.

5.6.2 GameplayKit

GameplayKit è una libreria introdotta da Apple con iOS 9 (2015). Come il nome suggerisce, è una libreria pensata principalmente per i videogiochi, ma che in realtà si presta ad essere usata anche in molti altri ambiti.

Non si tratta di una libreria per l'implementazione della parte grafica o della parte audio di un programma. Questa libreria è pensata per l'implementazione della parte logica delle App. Dato che questa libreria si occupa solo della parte logica, è necessario usare anche altre librerie per poter ottenere una App funzionante. Per questo motivo, Apple stessa suggerisce di usare GameplayKit insieme ad altre librerie, ad esempio SpriteKit [36].

In questo lavoro di tesi, si è scelto di usare GameplayKit principalmente per due motivi:

1) Include al suo interno le classi GKGraph e GKGraphNode pensate per la memorizzazione di un grafo e dei suoi nodi.
2) Include al suo interno delle funzionalità di Pathfinding che potrebbero essere utili per gli sviluppi futuri di questa App.

5.7 Implementazione della App

In questa sezione viene descritta la fase di implementazione e codifica della App.

5.7.1 Struttura della App

Questa App è stata scritta usando il linguaggio Objective-C. Il linguaggio Objective-C deriva dal linguaggio C (è un sovrainsieme del C), per questo motivo ogni classe viene realizzata tramite due file differenti: un file con la dichiarazione dell'interfaccia (file con estensione .h) ed un altro con la reale implementazione (file con estensione .m).

La seguente tabella mostra l'elenco delle classi principali che sono state realizzate per questo lavoro di tesi.

Nome classe	Descrizione
DAGrafo	Questa classe viene usata per rappresentare il grafo in memoria. E' una sottoclasse della classe GKGraph che fa parte di GameplayKit
DANodo	Questa classe è la base che è stata usata per rappresentare in memoria i nodi del grafo. E' una sottoclasse della classe GKGraphNode2D che fa parte di GameplayKit
DANodoArtista	Questa classe viene usata per memorizzare tutti i nodi del grafo che rappresentano artisti. E' una sottoclasse di DANodo
DANodoAlbum	Questa classe viene usata per memorizzare tutti i nodi del grafo che rappresentano album. E' una sottoclasse di DANodo
DANodoBrano	Questa classe viene usata per memorizzare tutti i nodi del grafo che rappresentano brani. E' un sottoclasse di DANodo
ViewController	Questa classe rappresenta il cuore dell'interfaccia della App come menù e pulsanti (UIKit) e contiene inoltre il player musicale.
DAScena01	Questa classe rappresenta visivamente il grafo. E' una sottoclasse della classe SKScene che fa parte di SpriteKit
DAPlaylistTableViewCell	Questa classe viene usata per rappresentare i risultati della ricerca per artista, album, brani. E' una sottoclasse di UITableViewCell
Reachability	Questa classe serve a testare la connessione ad internet. E' una sottoclasse di NSObject
DAReachability	Anche questa classe serve a testare la connessione ad internet. E' una sottoclasse di NSObject
costanti	Usata per contenere le più importanti costanti del progetto
DAUtilita	Usata per metodi comuni utili al progetto

La seguente tabella mostra l'elenco dei più importanti file aggiuntivi usati all'interno di questa App.

Nome file	Descrizione
Info.plist	Questo file è un Information Property List File che è presente in tutti i progetto di Xcode. Al suo interno sono specificate alcune chiavi che descrivono il comportamento della App, come ad esempio la necessità di accedere alla libreria musicale
Assets.xcassets	Questo file contiene tutti gli asset della App. In particolare contiene tutte le immagini usate dalla App (non contiene le copertine degli album che sono già presenti nella libreria musicale)
Main.storyboard	Questo file rappresenta la schermata principale della App con tutti gli elementi grafici inseriti direttamente tramite l'interfaccia di Xcode. Altri elementi grafici vengono poi generati ed inseriti direttamente nel codice
Scena01.sks	Questo file rappresenta la schermata grafica che contiene il grafo. Questo file viene caricato a Main.storyboard in modo che entrami siano contemporaneamente sullo schermo

Nelle prossime pagine entreremo nel dettaglio delle scelte implementative.

5.7.2 Implementazione del grafo

Per la creazione del grafo nel suo insieme in memoria si è scelto di usare la classe GKGraph che appartiene alla libreria GameplayKit. Per la rappresentazione del grafo sullo schermo si è scelto di usare la classe SKScene che appartiene allo SpriteKit Framework.

Questa scelta è stata determinata da diversi fattori:

1) Si è voluta separare la rappresentazione del grafo in memoria dalla rappresentazione dal grafo sullo schermo. La rappresentazione del grafo sullo schermo, infatti, viene realizzata tramite:
   - la classe SKScene per contenere il grafo
   - la classe SKSpriteNote per i nodi e per i simboli grafici di informazione all'utente
   - la classe SKShapeNode per gli archi
   - la classe SKLabelNode per i nomi degli artisti, degli album e dei brani

   Questo insieme di informazioni, sebbene necessario per la corretta visualizzazione su schermo, non consente di creare in memoria un grafo pulito della libreria musicale.

2) L'implementazione del grafo tramite GKGraph ha consentito l'uso della classe GKGraphNode2D di GameplayKit che si è rivelata molto adatta per la memorizzazione dei nodi del grafo (si veda la sezione successiva dedicata all'implementazione dei nodi).
3) La separazione del grafo in memoria dal grafo sullo schermo ha consentito di rendere la App molto più flessibile. E' stato ad esempio possibile creare e memorizzare i nodi connessi ai nodi sullo schermo, senza la necessità di doverli rappresentare su schermo prima del necessario. Questo tipo di ottimizzazione ha contribuito a rendere la App veloce e di basso impatto sull'uso della CPU e della batteria dell'iPad.

La seguente illustrazione mostra uno schema della classe GKGraph, dei suoi attributi e metodi principali e delle sue due sottoclassi [37].

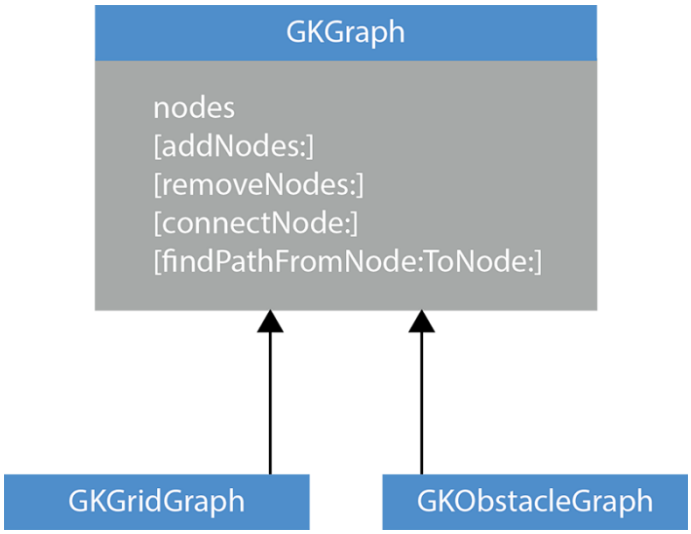


Fig. 26 La classe GKGraph

Nell'ambito di questo lavoro di tesi si è scelto di usare la classe principale GKGraph e non una delle sue sottoclassi in quanto si tratta di sottoclassi specializzate per usi non attinenti. GKGridGraph viene usata per grafi rappresentanti su griglie con posizioni definite da numeri interi (si pensi ad esempio ad una scacchiera). GKObstacleGraph viene invece usata per definire percorsi all'interno del grafo che evitano eventuali ostacoli appartenenti alla classe GKObstacle che sono stati inseriti nel grafo. La classe GKGraph non è stata utilizzata direttamente. E' stata realizzata una apposita sottoclasse chiamata DAGrafo.

Il codice del file interface DAGrafo.h creato per questo progetto è il seguente:

```
// DAGrafo.h

#import <GameplayKit/GameplayKit.h>
#import "DANodo.h"
#import "DANodoArtista.h"
#import "DANodoAlbum.h"
#import "DANodoBrano.h"
```

```
@interface DAGrafo : GKGraph

-(id)init;
-(GKGraphNode2D *)cercaNodoGrafoTipo:(NSString *)tipo nome:(NSString *)nome
identificativo:(NSString *)identificativo;

@end
```

Si tratta di un file implementazione breve dato che i metodi e le proprietà richieste per creare il grafo in memoria sono già contenute nella superclasse.

DAGrafo importa le classi relative ai vari tipi di nodi per poter implementare il metodo [cercaNodoGrafoTipo: nome: identificativo:] che viene utilizzato per trovare un nodo all'interno del grafo in base al nome ed al tipo.

La rappresentazione grafica del grafo è implementata nella classe DAScena01 che analizzeremo in dettaglio più avanti. DAScena01 è una sottoclasse di SKScene appartenente al framework SpriteKit.

5.7.3 Implementazione dei nodi

L'implementazione dei nodi, sia a livello di memoria sia su schermo, ha richiesto la creazione di nuove classi e lo sviluppo di metodi appositamente scritti per questo lavoro di tesi, dato che allo stato attuale non esistono librerie disponibili con tali funzionalità.

Per l'implementazione dei nodi in memoria, è stata creata la classe DANode, sottoclasse della classe GKGraphNode2D. A sua volta deriva GKGraphNode2D è una sottoclasse di GKGraphNode.

La seguente illustrazione mostra uno schema della classe GKGraphNode, dei suoi attributi e metodi principali e delle sue due sottoclassi.

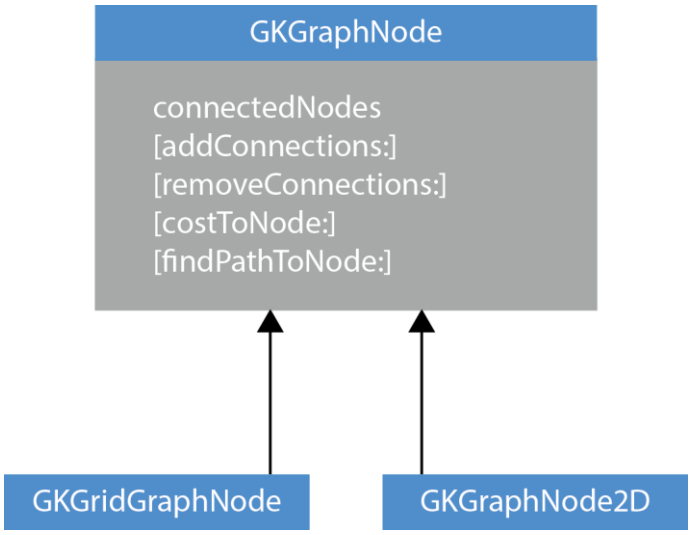


Fig. 27 La classe GKGraphNode

Nell'ambito di questo lavoro di tesi si è scelto di usare la classe GKGraphNode2D in quando, oltre ad ereditare gli attributi ed i metodi della sua superclasse, ha anche l'attributo position ed alcuni metodi che servono per la gestione della posizione del nodo nello spazio.

Il codice del file interface DANodo.h creato per questo progetto è il seguente:

```
//  DANodo.h

#import <GameplayKit/GameplayKit.h>
#import <SpriteKit/SpriteKit.h>
#import <MediaPlayer/MediaPlayer.h>
#import "costanti.h"

@interface DANodo : GKGraphNode2D

@property(nonatomic,strong) NSString *nome;
@property(nonatomic,strong) NSString *persistentID;
@property(nonatomic,assign) BOOL collegamentiDisponibili;
@property(nonatomic,assign) BOOL mostraIconaCollegamenti;

@property(nonatomic,strong) NSMutableArray *nodiCollegatiVisibile;
@property(nonatomic,strong) NSMutableArray *nodiCollegati;
@property(nonatomic,strong) NSMutableArray *nodiCollegatiTipoCollegamento;

@property(nonatomic,strong) SKSpriteNode *nodo;
@property(nonatomic,assign) BOOL nonAggiungereAlGrafo;

-(id)init;
-(void)aggiungiCollegamentiAiNodi:(NSMutableArray *)nodi tipoCollegamento:(NSString
*)tipoCollegamento;
-(BOOL)tuttiCollegamentiVisibili;

@end
```

E' importante ricordare che questa classe serve come base per la creazione di altre tre sottoclassi specifiche per la creazione dei nodi degli artisti, degli album e dei brani. Nel codice si notano la creazione di una serie di proprietà usate per memorizzare informazioni di base relative ai nodi.

classe DANodo	
Proprietà	Descrizione
nome	Viene usata per memorizzare il nome dell'elemento rappresentato dal nodo, ad esempio il nome dell'artista
persistentID	Rappresenta il numero identificativo univoco dell'elemento rappresentato dal nodo, ad esempio un artista
collegamentiDisponibili	Serve ad indicare che il nodo ha dei collegamenti, di qualsiasi tipo. Ad esempio se il nodo rappresenta un artista e la libreria contiene degli album di quel artista, questa proprietà avrà valore YES.
mostraIconaCollegamenti	Serve a stabilire se il nodo debba o meno mostrare un simbolo grafico che segnala la presenza di collegamenti da mostrare
nodiCollegati nodiCollegatiVisibile nodiCollegatiTipoCollegamento	Questi tre vettori vengono usati per memorizzare l'elenco dei collegamenti di questo nodo con altri nodi del grafo. Questi vettori vengono usati per tutti i collegamenti avanzati, vale a dire quelli non derivanti dalla relazione tra Artista, Album e Brano.
nodo	Viene usata per avere il riferimento al nodo nella sua implementazione grafica. Se nodo è nil significa che il nodo non è rappresentato su schermo
nonAggiungereAlGrafo	Usata per evitare di inserire due volte lo stesso nodo nel grafo

La classe DANodo dispone inoltre di tre metodi usati per la gestione del nodo stesso.

classe DANodo	
Metodo	Descrizione
[init]	Usato per l'inizializzazione
[aggiungiCollegamentiAiNodi: tipoCollegamento:]	Inserisce nel nodo una serie di collegamenti avanzati. Questo metodo inserisce i collegamenti anche nel nodo collegato.
[tuttiCollegamentiVisibili]	Questo metodo verifica se tutti i collegamenti avanzati del nodo sono già visibili sullo schermo

La classe DANodo non viene normalmente usata direttamente, ma vengono usate le sue tre sottoclassi. Queste sottoclassi si sono rese necessarie perché le informazioni da memorizzare relativamente ad un artista, un album o un brano sono almeno in parte diverse.

Per rappresentare gli artisti, è stata creata la classe DANodoArtista, sottoclasse di DANodo. Il codice del file interface DANodoArtista.h creato per questo progetto è il seguente:

```
// DANodoArtista.h

#import <GameplayKit/GameplayKit.h>
#import <SpriteKit/SpriteKit.h>
#import "DANodo.h"

@interface DANodoArtista : DANodo

@property(nonatomic,strong) NSString *resourceUrl;
@property(nonatomic,assign) BOOL tuttiGliAlbumAperti;

-(id)init;
-(id)initConDati:(MPMediaItemCollection *)artista;

@end
```

Questa classe implementa poche proprietà aggiuntive rispetto alla sua superclasse.

classe DANodoArtista	
Proprietà	Descrizione
resourceUrl	Viene usata per memorizzare l'indirizzo web del sito Discogs che contiene informazioni su questo artista
tuttiGliAlbumAperti	Viene usata per segnalare che tutti gli album di questo artista sono già stati mostrati sullo schermo

La classe DANodoArtista dispone inoltre di due metodi usati per la gestione del nodo stesso.

classe DANodoArtista	
Metodo	Descrizione
[init]	Usato per l'inizializzazione di un nuovo artista vuoto
[initConDati:]	Usato per inizializzare il nodo dell'artista con le informazioni lette dalla music library dell'iPad

Per rappresentare gli album, è stata creata la classe DANodoAlbum, sottoclasse di DANodo. Il codice del file interface DANodoAlbum.h creato per questo progetto è il seguente:

```
// DANodoAlbum.h

#import <GameplayKit/GameplayKit.h>
#import <SpriteKit/SpriteKit.h>
#import "DANodo.h"
#import "costanti.h"

@interface DANodoAlbum : DANodo

@property(nonatomic,strong) NSString *nomeArtista;
@property(nonatomic,strong) NSString *persistentIDArtista;
@property(nonatomic,strong) NSString *nomeArtistaAlbum;
@property(nonatomic,strong) NSString *persistentIDArtistaAlbum;
@property(nonatomic,strong) NSNumber *annoAlbum;
```

```
-(id)init;
-(id)initConDati:(MPMediaItemCollection *)album;
-(id)initConArtistaAlbum:(NSString *)IDArtistaAlbum IDAlbum:(NSString *)IDAlbum;

@end
```

Questa classe implementa diverse proprietà aggiuntive rispetto alla sua superclasse.

classe DANodoAlbum	
Proprietà	Descrizione
nomeArtista	Nome dell'artista
persistentIDArtista	Rappresenta il numero identificativo univoco dell'artista
nomeArtistaAlbum	Nome dell'artista dell'album
persistentIDArtistaAlbum	Rappresenta il numero identificativo univoco dell'artista dell'album
annoAlbum	Anno di pubblicazione dell'album

In riferimento a queste proprietà si segnala la presenza sia della proprietà "nomeArtista" sia della proprietà "nomeArtistaAlbum". Queste informazioni non coincidono sempre. Una più estesa spiegazione verrà data tra poche pagine, nello spazio dedicata alla classe DANodoBrano.

In riferimento alla proprietà annoAlbum, si segnala che questa informazione non è facilmente reperibile nella libreria musicale dell'iPad. Per accedere a questa informazione è stato necessario usare una proprietà non documentata della classe MPMediaItem.

La classe DANodoAlbum dispone inoltre di tre metodi usati per la gestione del nodo stesso.

classe DANodoAlbum	
Metodo	Descrizione
[init]	Usato per l'inizializzazione di un nuovo album vuoto
[initConDati:]	Usato per inizializzare il nodo dell'album con le informazioni lette dalla music library dell'iPad
[initConArtistaAlbum: IDAlbum:]	Usato per inizializzare il nodo dell'album con le informazioni lette dalla music library dell'iPad. Rispetto al precedente usa diversi argomenti.

Per rappresentare i brani, è stata creata la classe DANodoBrano, sottoclasse di DANodo. Il codice del file interface DANodoBrano.h creato per questo progetto è il seguente:

```
//  DANodoBrano.h

#import <GameplayKit/GameplayKit.h>
#import <SpriteKit/SpriteKit.h>
#import <MediaPlayer/MediaPlayer.h>
#import "DANodo.h"
#import "costanti.h"

@interface DANodoBrano : DANodo

@property(nonatomic,strong) NSString *tracciaNumero;
@property(nonatomic,strong) NSString *tracceAlbum;
@property(nonatomic,strong) NSURL *urlBrano;
@property(nonatomic,strong) UIImage *copertina;
@property(nonatomic,strong) NSString *nomeArtista;
@property(nonatomic,strong) NSString *persistentIDArtista;
@property(nonatomic,strong) NSString *nomeArtistaAlbum;
@property(nonatomic,strong) NSString *persistentIDArtistaAlbum;
@property(nonatomic,strong) NSString *nomeAlbum;
@property(nonatomic,strong) NSString *persistentIDAlbum;
@property(nonatomic,strong) NSString *statoBrano;
@property(nonatomic,strong) NSString *nomeCompositore;
@property(nonatomic,strong) NSString *persistentIDCompositore;
@property(nonatomic,strong) NSString *durata;
@property(nonatomic,strong) NSNumber *annoBrano;

-(id)init;
-(id)initConDati:(MPMediaItem *)brano;

@end
```

Questa classe implementa molte proprietà aggiuntive rispetto alla sua superclasse.

classe DANodoBrano	
Proprietà	Descrizione
tracciaNumero	Rappresenta il numero del brano all'interno dell'album che lo contiene
tracceAlbum	Rappresenta il numero di brani dell'album
urlBrano	Rappresenta il file del brano all'interno della music library
copertina	E' la copertina del brano
nomeArtista	E' il nome dell'artista del brano
persistentIDArtista	Rappresenta il numero identificativo univoco dell'artista
nomeArtistaAlbum	E' il nome dell'artista dell'album
persistentIDArtistaAlbum	Rappresenta il numero identificativo univoco dell'artista dell'album

nomeAlbum	E' il titolo dell'album
persistentIDAlbum	Rappresenta il numero identificativo univoco dell'album
statoBrano	Indica se il brano è attualmente nella playlist, oppure è in esecuzione.
nomeCompositore	E' il nome del compositore
persistentIDCompositore	Rappresenta il numero identificativo univoco del compositore
durata	Durata del brano in secondi e frazioni di secondo.
annoBrano	Anno di pubblicazione del brano

In riferimento a queste proprietà si segnala la presenza sia della proprietà "nomeArtista" sia della proprietà "nomeArtistaAlbum". Queste informazioni non coincidono sempre. Si pensi ad esempio ad un album di un artista che contiene un brano cantato insieme ad un altro artista. In questo specifico brano, la proprietà nomeArtista sarà l'insieme dei due nome. La proprietà nomeArtistaAlbum sarà invece solo il nome di un artista. La presenza di questi due campi nella libreria musicale dell'iPad può creare situazioni poco chiare quando questi campi (metadati) non vengono compilati correttamente.

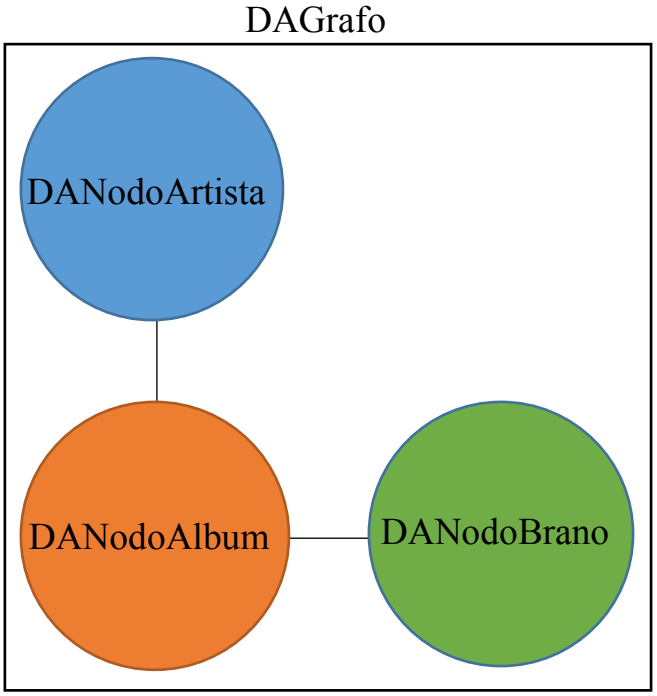
Anche per la proprietà annoBrano, si segnala che questa informazione non è facilmente reperibile nella libreria musicale dell'iPad. Per accedere a questa informazione è stato necessario usare una proprietà non documentata della classe MPMediaItem.

La classe DANodoBrano dispone inoltre di due metodi usati per la gestione del nodo stesso.

classe DANodoBrano	
Metodo	Descrizione
[init]	Usato per l'inizializzazione di un nuovo brano vuoto
[initConDati:]	Usato per inizializzare il nodo del brano con le informazioni lette dalla music library dell'iPad

Quindi, a titolo di esempio, la rappresentazione in memoria del grafo con un artista, un album ed un brano avviene tramite l'uso delle classi: DAGrafo, DANodoArtista, DANodoAlbum e DANodoBrano.

La seguente illustrazione mostra una schematizzazione dell'uso di queste quattro classi.



Per rappresentare il grafo su schermo è stata creata la classe DAScena01, sottoclasse della classe SKScene che appartiene al framework SpriteKit. Il codice del file interface DAScena01.h creato per questo progetto è il seguente:

```
//  DAScena01.h

#import <SpriteKit/SpriteKit.h>
#import <MediaPlayer/MediaPlayer.h>
#import "DAShortTapGestureRecognizer.h"
#import "DAGrafo.h"
#import "DANodoArtista.h"
#import "DANodoAlbum.h"
#import "DANodoBrano.h"
#import "costanti.h"
#import "DAReachability.h"

@interface DAScena01 : SKScene<SKPhysicsContactDelegate>

@property(nonatomic,strong) DAGrafo *grafo;

-(void)inserisciArtista:(DANodoArtista *)artistaDaCreare artistaGiaNelGrafo:(BOOL)artistaGiaNelGrafo;
-(void)inserisciSoloArtista:(DANodoArtista *)artistaDaCreare;
-(void)cambiaStatoBrano:(DANodoBrano *)brano nuovoStato:(NSUInteger)nuovoStatoBrano;
-(NSMutableArray *)cercaAlbumArtista:(NSString *)nome identificativo:(NSString *)idArtista;
-(NSUInteger)quantiAlbumArtista:(NSString *)nome identificativo:(NSString *)idArtista;
-(void)disegnaGrafo:(NSMutableArray *)nodiDaDisegnare;
-(MPMediaItemArtwork *)trovaCopertinaAlbum:(NSString *)nomeAlbum identificativo:(NSString
*)idAlbum;
-(UIImage *)ridimensionaImmagine:(UIImage *)originale nuovaDimensione:(CGSize)dimensione;
-(void)zoomSuNodo:(SKSpriteNode *)nodo;

-(NSMutableArray *)cercaBraniAlbum:(NSString *)nome identificativo:(NSString *)idAlbum;
```

```
-(BOOL)cercaBraniCover:(DANodoBrano *)brano;
-(BOOL)cercaBraniIdentici:(DANodoBrano *)brano;
-(BOOL)cercaCollegamentiArtista:(DANodoArtista *)artista;
-(void)verificaCollegamentiArtista:(DANodoArtista *)artista collegamentiDaVerificare:(NSMutableArray
*)collegamentiDaVerificare;
-(DANodoArtista *)cercaArtista:(NSString *)nome;
-(void)inserisciAlbum:(DANodoAlbum *)albumDaCreare albumGiaNelGrafo:(BOOL)albumGiaNelGrafo;
-(void)inserisciSoloAlbum:(DANodoAlbum *)albumDaCreare;
-(void)inserisciSoloBrano:(DANodoBrano *)branoDaCreare;
-(void)mostraCollegamentiBrano:(DANodoBrano *)brano branoGiaNelGrafo:(BOOL)branoGiaNelGrafo;
-(void)mostraCollegamentiArtista:(DANodoArtista *)artista artistaGiaNelGrafo:(BOOL)artistaGiaNelGrafo;
-(void)aggiornaIconaCollegamenti:(GKGraphNode2D *)nodo;
-(void)aggiornaIconaProblemiConnessione:(GKGraphNode2D *)nodo mostra:(BOOL)mostra;
-(void)aggiornaEstensioneMaxNodi;
-(vector_float2)posizioneNuovoNodo:(float)raggioNuovoNodo;
-(void)confrontaPosizioneX:(float)x posizioneY:(float)y;

-(NSMutableArray *)cercaBraniAlbumDaBrano:(SKSpriteNode *)brano;
-(void)ridisegnaArchi:(SKSpriteNode *)nodo chiamatoDaNodo:(SKSpriteNode *)nodoOriginale;
-(void)chiamataRicorsivaRidisegnaArchi:(SKSpriteNode *)nodo;
-(void)eliminaTutto;

@end
```

Questa classe implementa una sola proprietà aggiuntiva rispetto alla sua superclasse.

classe DAScena01	
Proprietà	Descrizione
grafo	Rappresenta il grafo in memoria

In questa sede non analizzeremo tutti i metodi di questa classe, ma solo quelli rilevanti al fine della visualizzazione del grafo sullo schermo.

classe DAScena01	
Metodo (selezione)	Descrizione
[inserisciArtista:artistaGiaNelGrafo:]	Usato per eseguire le seguenti operazioni: - Inserire un artista e tutti i suoi album nel grafo - Mostrare su schermo un artista e tutti i suoi album
[inserisciSoloArtista:]	Usato per eseguire le seguenti operazioni: - Inserire un artista nel grafo - Mostrare su schermo un artista
[inserisciAlbum: albumGiaNelGrafo:]	Usato per eseguire le seguenti operazioni: - Inserire un album e tutti i suoi brani nel grafo - Mostrare su schermo un album e tutti i suoi brani
[inserisciSoloAlbum:]	Usato per eseguire le seguenti operazioni: - Inserire un album nel grafo - Mostrare su schermo un album

[inserisciSoloBrano:]	Usato per eseguire le seguenti operazioni: - Inserire un brano nel grafo - Mostrare su schermo un artista
[disegnaGrafo:]	Questo metodo serve a disegnare nuovi nodi del grafo su schermo. Questo metodo non viene usato per aggiornare il grafo in caso di spostamento di nodi sullo schermo.

I primi cinque metodi elencati in tabella servono per l'inserimento e la rappresentazione su schermo, rispettivamente, di artisti, album e brani musicali. Tutti questi metodi usano il metodo [disegnaGrafo:] per disegnare nuovi nodi del grafo su schermo.

Il metodo [disegnaGrafo:] è troppo lungo per inserirlo in forma integrale in queste pagine, ma è importante sapere che serve per una di queste operazioni:

1) Rappresentare su schermo un artista
2) Rappresentare su schermo un artista ed i suoi album
3) Rappresentare su schermo un album
4) Rappresentare su schermo un album ed i suoi brani
5) Rappresentare su schermo un brano

Per eseguire queste operazioni il metodo usa principalmente due variabili:

```
SKSpriteNode *nodoPrincipale;
SKSpriteNode *nodoSecondario;
```

La prima, nodoPrincipale, rappresenta nei casi 1,2,3 appena visti il nodo dell'artista mentre nei casi 4 e 5 rappresenta l'album. La seconda variabile, nodoSecondario, rappresenta dei casi 1,2,3 l'album mentre nei casi 4 e 5 rappresenta un brano.

Come anticipato nella sezione dedicata a SpriteKit, il modo di inserire su schermo i nodi è quello di creare un albero la cui radice è la classe DAScena01 (sottoclasse di SKScene). Bisogna poi inserire nell'albero nodoPrincipale che appartiene ad una sottoclasse di SKNode, nel nostro caso SKSpriteNode. Bisogna quindi inserire i figli di nodoPrincipale, vale a dire nodoSecondario.

Le classi che sono state utilizzate per la rappresentazione degli elementi su schermo sono elencate nella tabella seguente:

DAScena01	
Elemento su schermo	Classe di SpriteKit utilizzata
Artista	SKSpriteNode
Nome dell'artista	SKLabelNode

Icona collegamenti artista	SKSpriteNode
Album	SKSpriteNode
Linea tra album e artista	SKShapeNode
Nome dell'album	SKLabelNode
Icona collegamenti album	SKSpriteNode
Brano	SKSpriteNode
Linea tra brano e album	SKShapeNode
Nome del brano	SKLabelNode
Icona collegamenti brano	SKSpriteNode

La rappresentazione su schermo di un artista, un album ed un brano comporta la creazione dell'albero mostrato in figura:

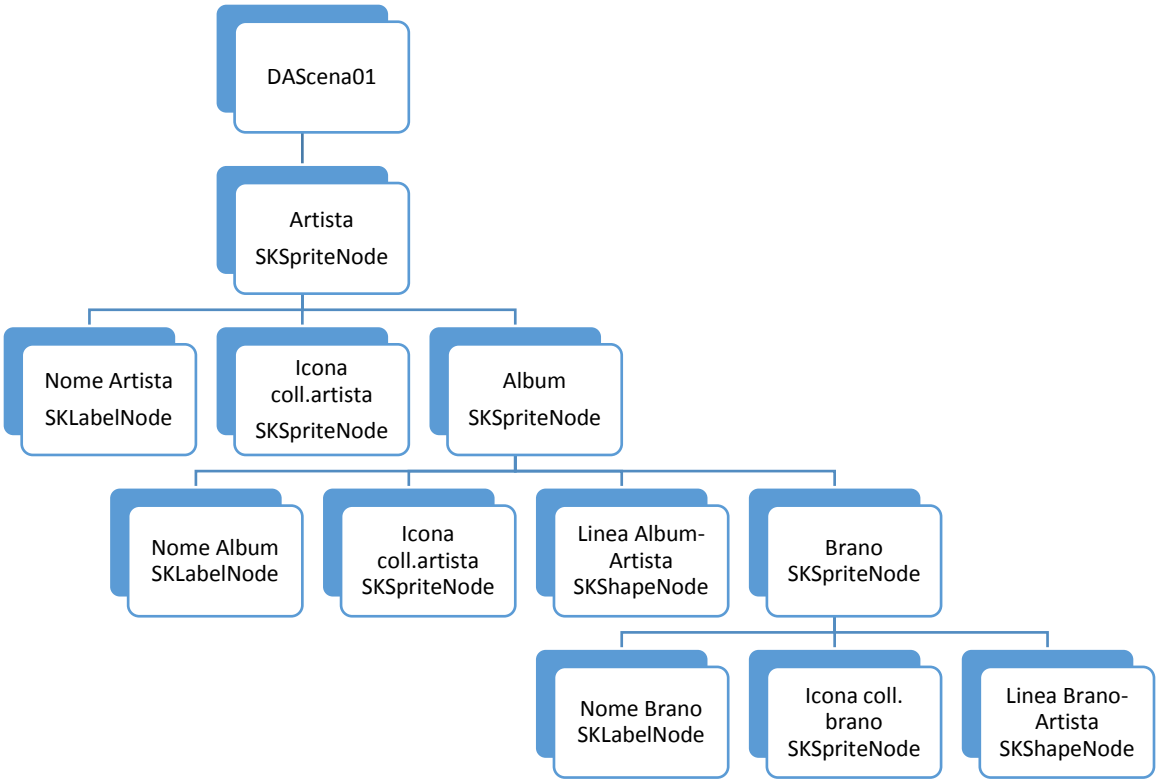


Fig. 28 Albero per rappresentare su schermo un artista, un album ed un brano

Sottolineiamo che lo sviluppo di questa App ha richiesto la realizzazione di una libreria appositamente scritta per la rappresentazione dei grafi, dato che allo stato attuale non esistono librerie disponibili con tali funzionalità. La maggior parte del codice si trova all'interno della classe DAScena01.

5.7.4 Implementazione dei collegamenti avanzati tra nodi

Oltre ai collegamenti normalmente esistenti tra un artista, i suoi album ed i suoi brani, la App consente di visualizzare dei collegamenti avanzati normalmente non presenti dei player musicali in commercio.

I collegamenti avanzati che si è scelto di implementare sono: ricerca di brani cover o comunque versioni alternative, ricerca di brani identici, ricerca di collegamenti tra artisti del tipo gruppo musicale-membro della band.

Nella seguente tabella vengono indicati i parametri principali usati per la ricerca dei collegamenti avanzati.

Collegamenti avanzati tra nodi	
Tipo	Parametri
Cover o versioni alternative	- Titolo uguale - Stesso compositore
Brani identici	- Titolo uguale - Stesso artista - Stessa durata ± 4 secondi
Collegamenti tra artisti (gruppo-membro)	- Ricerca in base al nome nella banca dati online Discogs

In particolare per quanto riguarda il collegamento tra artisti, ogni volta che il nodo di un artista viene mostrato sullo schermo, la App verifica, se nella libreria online Discogs esiste un artista con lo stesso nome. Le API di Discogs rispondono usando il formato JSON. Un volta trovato l'artista, la App esegue una seconda interrogazione a Discogs per aprire la pagina specifica dell'artista. Dentro a questa pagina, anch'essa ricevuta dalla App in formato JSON, viene effettuata la ricerca per capire se l'artista ha dei collegamenti con altri artisti.

Per la ricerca di collegamenti tra artisti vengono utilizzati i seguenti metadati di Discogs:

Proprietà	Significato
title	Il nome dell'artista
members	Elenco di membri di un gruppo. Questa proprietà è vuota se si tratta di un artista
groups	Elenco dei gruppi nei quali l'artista ha lavorato

I metodi all'interno di DAScena01 usati per la gestione dei collegamenti avanzati tra nodi sono riportati nella seguente tabella.

classe DAScena01	
Metodo (selezione)	Descrizione
[cercaBraniCover:]	Usato per cercare le cover e le versioni alternative di un brano
[cercaBraniIdentici:]	Usato per cercare i brani identici
[cercaCollegamentiArtista:]	Usato per iniziare la ricerca dei collegamenti tra artisti
[verificaCollegamentiArtista: collegamentiDaVerificare:]	Usato per verificare quale degli artisti collegati segnalati da Discogs sono effettivamente presenti nella libreria musicale dell'iPad
[mostraCollegamentiBrano: branoGiaNelGrafo:]	Usato per mostrare su schermo i collegamenti tra brani
[mostraCollegamentiArtista: artistaGiaNelGrafo:]	Usato per mostrare su schermo i collegamenti tra artisti

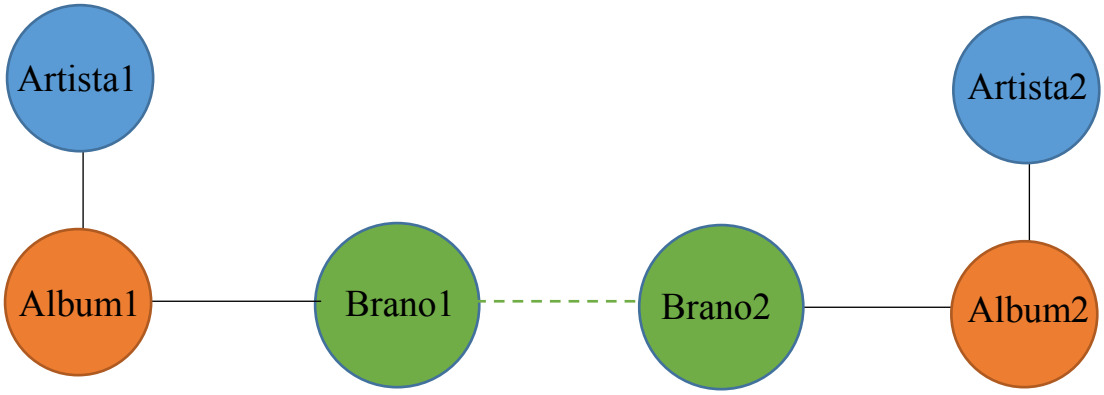
Le linee di collegamento tra brani e tra artisti vengono realizzate tramite la classe SKShapeNode che appartiene al framework SpriteKit. Per queste linee viene sfruttata una funzionalità aggiuntiva di questa classe che consente di memorizzare all'interno dell'attributo userData una serie di informazioni scelte del programmatore. Le informazioni aggiuntive che sono state memorizzate sono:

- daNodo
- aNodo
- tratteggio

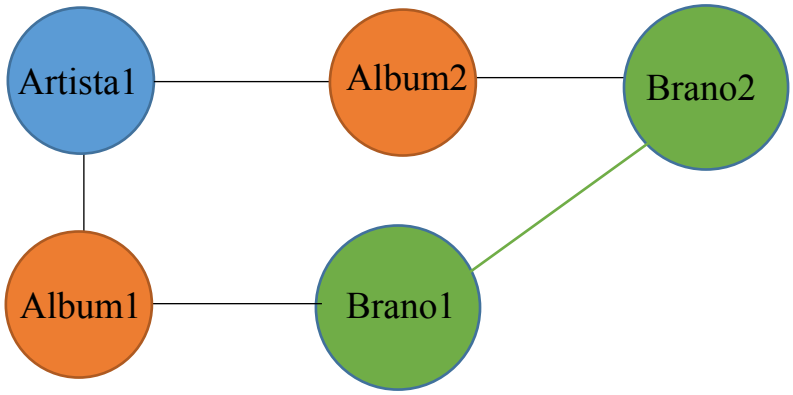
Queste informazioni vengono usate per disegnare la linea e per aggiornare lo schermo quando l'utente decide di muovere uno o più nodi sullo schermo. Si veda la sezione 5.5.3 per una dettagliata descrizione della rappresentazione di questi collegamenti.

Riepiloghiamo brevemente il tipo di rappresentazione usato per i collegamenti avanzati:

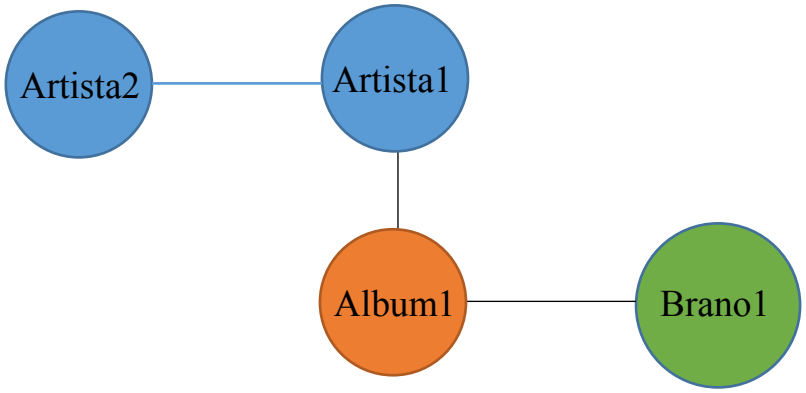
1) Il collegamento tra i brani cover avviene tramite una linea tratteggiata di colore verde (come i nodi dei brani).



2) Il collegamento tra i brani identici avviene tramite una linea continua di colore verde (come i nodi dei brani).



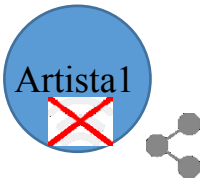
3) Il collegamento tra artisti avviene tramite una linea continua di colore blu (come i nodi degli artisti).



L'implementazione della ricerca di informazioni sul web ha comportato la risoluzione di una serie di problemi legati sia al collegamento stesso alla rete, sia alla corrispondenza tra le informazioni trovate in internet con quelle disponibili nella libreria musicale dell'iPad.

Per quanto concerne il collegamento ad Internet, prima di effettuare una qualsiasi interrogazione in Internet è necessario verificare se il collegamento ad Internet è disponibile oppure no. Questa verifica deve essere effettuata prima di ogni interrogazione. Il motivo è che la connessione ad internet dei dispositivi mobili come l'iPad è potenzialmente molto instabile. E' infatti possibile che in un dato istante la connessione ci sia e che pochi secondi dopo la connessione non sia più disponibile (si pensi, ad esempio, ad una vettura che entra in una galleria).

Nel caso in cui, invece, il collegamento ad Internet non è disponibile, la App mostra un apposito simbolo sul nodo dell'artista per indicare che durante l'ultimo tentativo di reperire informazioni su quell'artista c'è stato un problema. La App proverà ad effettuare un nuovo collegamento ad Internet ogni volta che viene toccato il nodo dell'artista.



Una volta verificata la presenza della connessione, il metodo [cercaCollegamentiArtista:] fa un tentativo di connessione alla banca dati online Discogs. L'interrogazione che viene fatta serve a trovare su Discogs l'artista della libreria musicale dell'iPad che si sta cercando. Una volta ricevuta risposta positiva (l'artista è stato trovato), il metodo [cercaCollegamentiArtista:] fa una seconda interrogazione a Discogs, questa volta per aprire la pagina dedicata all'artista. Dentro a questa pagina vengono analizzati i campi (metadati) groups oppure members. Questi campi contengono, potenzialmente, degli elenchi. Questi elenchi vengono poi analizzati dal metodo [verificaCollegamentiArtista: collegamentiDaVerificare:] che cerca corrispondenza tra i nomi degli artisti trovati su Discogs e quelli presenti nella libreria dell'iPad. E' infatti possibile che l'elenco riportato da Discogs includa dei nomi non presenti nell'iPad. In questo casi i nomi in eccesso vengono scartati.

E' importante sottolineare un altro punto relativo al collegamento ad Internet della App. Il collegamento ad Internet, oltre ad essere instabile, è anche potenzialmente lento. Il tempo di risposta all'interrogazione fatta sul web non è dato. La risposta potrebbe anche arrivare dopo alcuni secondi. Per questo motivo è fondamentale che l'esecuzione di queste interrogazioni sul web venga fatto in un thread differente da quello principale della App. Se questo non viene fatto il rischio è che la App rimanga bloccata fino a quando arriva la risposta da Internet.

5.7.5 Implementazione delle interazioni utente-grafo

Si è deciso di implementare una App che permette un'ampia interazione tra l'utente ed il grafo rappresentato su schermo.

L'utente è infatti in grado di muovere tutto il grafo, muovere un singolo nodo, muovere un gruppo di nodi, effettuare il pinch to zoom, espandere i nodi ed avviare la riproduzione musicale.

L'implementazione delle interazioni utente-grafo si trova all'interno della classe DAScena01, che è una sottoclasse di SKScene. SKScene fa parte del framework SpriteKit.

La seguente tabella contiene l'elenco delle interazioni e dei rispettivi metodi.

Interazione	Metodo
Muovere il grafo	[gestisciPanGesture:]
Muovere un singolo nodo	[touchesMoved: withEvent:] [panForTranslation: numeroDita:]
Muovere un gruppo di nodi	[touchesMoved: withEvent:] [panForTranslation: numeroDita:]
Pinch to zoom	[gestisciZoomFrom:]
Tap	[gestisciTap:]
Doppio tap	[gestisciDoubleTap:]

Di queste interazioni, alcune richiedono una maggior precisione nella lettura dei gesti effettuati dall'utente. Si tratta delle interazioni che devono riconoscere con esattezza il nodo con il quale l'utente ha deciso di interagire. Il codice per il riconoscimento del nodo scelto dall'utente è il seguente:

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    for (UITouch *t in touches) {
        CGPoint location = [t locationInNode:self];
        [self scegliNodoPerTocco:location];
    }

}
-(void)scegliNodoPerTocco:(CGPoint)touchLocation{
    SKSpriteNode *touchedNode=(SKSpriteNode *)[self nodeAtPoint:touchLocation];

    if([touchedNode.name isEqualToString:@"etichetta"]){
        _selectedNode=(SKSpriteNode *)touchedNode.parent;
    } else if([touchedNode.name isEqualToString:@"linea"]){
        _selectedNode=(SKSpriteNode *)touchedNode.parent;
    } else if([touchedNode.name isEqualToString:@"collegamenti"]){
        _selectedNode=(SKSpriteNode *)touchedNode.parent;
    } else if([touchedNode.name isEqualToString:@"stato"]){
        _selectedNode=(SKSpriteNode *)touchedNode.parent;
```

```
    } else if([touchedNode.name isEqualToString:@"connessione_problemi"]){
        _selectedNode=(SKSpriteNode *)touchedNode.parent;
    } else {
        _selectedNode=touchedNode;
    }
    if(_selectedNode.name==nil){
        // NESSUN NODO TOCCATO
        _selectedNode=nil;
    }
}
```

Il nodo scelto dall'utente viene memorizzato nella variabile selectedNode di tipo SKSpriteNode.

Nel codice appena mostrato risulta chiaro che tutte le volte che l'utente tocca non direttamente il nodo ma uno dei suoi figli (ad esempio l'etichetta con il nome), il metodo [scegliNodoPerTocco:] deve risalire al nodo padre (non sarebbe corretto muovere il nome di un nodo senza il nodo).

Tutte le volte che l'utente muove un nodo sullo schermo, vengono richiamati due metodi:

classe DAScena01	
Metodo (selezione)	Descrizione
[ridisegnaArchi: chiamatoDaNodo:]	Usato per cancellare e ridisegnare i lati (o archi) collegati al nodo durante il trascinamento
[ridisegnaEtichetta]	Usato per aggiornare l'etichetta con il nome del nodo durante il trascinamento

Questi due metodi vengono richiamati non all'inizio ed alla fine del trascinamento, ma per tutta la durata dello stesso. Ogni volta che iOS comunica tramite il metodo [touchesMoved: withEvent:] che la posizione del tocco sullo schermo si è spostata, la App richiama i due metodi elencati in tabella. Questo consente di avere un'esperienza utente appagante, dato che l'utente non si rende conto del fatto che, ad esempio, [ridisegnaArchi: chiamatoDaNodo:] sta continuamente cancellando e disegnando gli archi di collegamento tra i nodi.

Ci sono due casi notevoli nello spostamento dei nodi sullo schermo:

1) Quando l'utente vuole muovere, ad esempio, un album e tutti i suoi brani l'implementazione è relativamente semplice dato che SpriteKit muove insieme al nodo scelto dall'utente (l'album) anche tutti i nodi che si trovano nell'albero dei figli di quel nodo (vedi capitolo 5.7.3). Quindi i metodi [ridisegnaArchi: chiamatoDaNodo:] e [ridisegnaEtichetta] vengono chiamati per nodo mosso dall'utente. Il metodo [ridisegnaArchi: chiamatoDaNodo:] viene chiamato anche per tutti i figli del nodo mosso dall'utente. Questo per gestire i collegamenti avanzati.

2) Quando l'utente muove un nodo che ha dei collegamenti avanzati (come ad esempio un brano che ha delle cover nella libreria) il metodo [ridisegnaArchi: chiamatoDaNodo:] viene chiamato non solo per i nodi visti nel punto 1, ma anche per tutti i nodi con collegamento avanzato collegati al nodo. L'elenco di questi nodi è contenuto nel vettore nodiCollegatiVisibile legato al nodo.

Il metodo [ridisegnaArchi: chiamatoDaNodo:] viene quindi chiamato molto frequentemente. Durante lo spostamento dei nodi. Nonostante questo, la App solitamente non scende mai sotto i 60 fps che vengono generati dalla scena DAScena01.

Il gesto di trascinamento, attivato muovendo due dita sullo schermo, muove sullo schermo tutto il grafo. Questo tipo di traslazione è attuato muovendo non il grafo ma muovendo la telecamera associata a DAScena01. Si tratta della variabile cameraPrincipale che appartiene alla classe SKCameraNode.

Anche il gesto pinch to zoom, attivato muovendo con due dita ma con un movimento di avvicinamento o allontanamento tra le dita, è stato realizzato muovendo non il grafo ma applicando un cambiamento di scala ed un cambiamento di posizione alla telecamera associata a DAScena01.

La gestione del double tap (doppio tocco veloce) introduce un certo delay (ritardo) nella gestione del tap (singolo tocco sullo schermo). iOS solitamente impiega 0,5 secondi per stabilire se si tratta di un tap o di un double tap. Durante lo sviluppo di questo lavoro di tesi, si è deciso che questo delay fosse troppo lungo. Per questo motivo è stata creata la classe DAShortTapGestureRecognizer (sottoclasse di UITapGestureRecognizer) che ha permesso di ridurre questo tempo di delay a 0,25 secondi.

5.7.6 Implementazione del player musicale

Il player musicale, vale a dire la parte del codice che si occupa direttamente della riproduzione audio dei brani, è stato implementato all'interno della classe ViewController.

L'argomento della AVAudioSession è già stato trattato nel capitolo 5.4.1. In questa sede ribadiamo i passaggi più importanti ai fini dell'implementazione.

E' opportuno che l'utente della App abbia un'esperienza consistente e prevedibile quando usa la App. Per questo motivo esistono le Audio Session di iOS. Tramite AVAudioSession la App comunica ad iOS che tipo di utilizzo la App fa dell'audio. Ad ogni applicazione che riproduce o registra audio in iOS viene automaticamente associata una Audio Session con dei valori di default.

Questi valori di default, però, non sono adatti per un'applicazione come quella che è oggetto di questo lavoro di tesi.

Per impostare una AVAudioSession bisogna seguire cinque importanti passaggi:

1) Ottenere un riferimento alla AVAudioSession della App.

Il codice è il seguente:

AVAudioSession *session=[AVAudioSession sharedInstance];

2) Impostare categoria, modo ed eventuali opzioni della AVAudioSession.

Esistono attualmente sette categorie tra le quali scegliere. La categoria pensata per un player musicale è AVAudioSessionCategoryPlayback.

3) Attivare la sessione.

   Senza questo comando le impostazioni che si sono scelte non vengono applicate. Una volta attivata la sessione, in base alle impostazioni che si sono scelte, la Audio Session determina l'interazione tra questa App e le altre App.

   Il codice per attivare la sessione è il seguente:

```
NSError *setActiveError=nil;
  success=[session setActive:YES error:&setActiveError];
 if(!success)
  {
 NSLog(@"ERRORE: setActive");
 }
```

4) Gestire le interruzioni.

   La sessione audio può essere interrotta, ad esempio da audio con priorità più alta: un tipico esempio è il suono di allarme del timer. L'interruzione rende la sessione inattiva ed interrompe l'audio ancora prima che la App riceva la notifica. Quando l'interruzione è finita bisogna gestire la situazione in modo appropriato, ad esempio riattivando la sessione audio e facendo ripartire l'audio [27].

   Per gestire le interruzioni bisogna registrare la App per ricevere le notifiche.

   Il codice è il seguente:

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(gestioneAudioSessionInterruption:) name:AVAudioSessionInterruptionNotification
object:session];
   [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(gestioneMediaServicesReset:)
name:AVAudioSessionMediaServicesWereResetNotification object:session];
```

E' importante notare che non tutte le interruzioni che hanno un inizio (TypeBegan) hanno anche una fine (TypeEnded).

5) Gestire i cambiamenti di periferiche audio (Route Changes).

   Tutti gli smartphone e tablet ed in particolare quelli basati sul sistema operativo iOS seguono delle consuetudini nella gestione delle periferiche audio che ogni App dovrebbe rispettare.

   Ad esempio, quando un utente che sta ascoltando della musica collega le cuffie al suo dispositivo, l'utente si aspetta che l'audio, che prima veniva riprodotto tramite gli altoparlanti del proprio device, venga ora riprodotto dalle cuffie e che la musica continui. Al contrario quando l'utente scollega le cuffie si aspetta che l'audio venga interrotto.

   Per gestire questo tipo di variazioni, è necessario configurare la App per ricevere queste notifiche.

   Il codice è il seguente:

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(gestioneRouteChange:)
name:AVAudioSessionRouteChangeNotification object:session];
```

Il framework audio che è stato usato in questo lavoro di tesi è AVAudioEngine. Un'approfondita analisi di questo framework può essere trovata nel capitolo 5.4.2.

AVAudioEngine, introdotta in iOS 8 e poi ulteriormente sviluppata nelle versioni successive di iOS, è nata con l'intenzione di fornire agli sviluppatori la possibilità di eseguire funzioni complesse in modo semplice.

L'elemento principale di questa API è la classe AVAudioEngine. Una istanza di questa classe rappresenta un contenitore che verrà poi utilizzato per collegare i nodi della catena di gestione del flusso audio. AVAudioEngine gestisce un grafo di nodi audio e consente una gestione dinamica del collegamento tra i nodi. I nodi, che sono sottoclassi di AVAudioNode, che sono stati usati in questo lavoro di tesi sono riportati nella seguente tabella.

Classe	Descrizione
AVAudioEngine	E' l'elemento principale che contiene i nodi della catena di gestione del flusso audio
AVAudioMixerNode	Usato per gestire il mixer principale di AVAudioEngine
AVAudioPlayerNode	Usato per la riproduzione dei file audio dalla libreria musicale
AVAudioFile	Usato per rappresentare il file audio della libreria musicale
AVAudioUnitEQ	Usato per creare un equalizzatore

AVAudioMixingDestination	Usato per gestire il flusso audio in uscita dal player

Il metodo principale per l'implementazione del player musicale è [eseguiBrano:]. Questo metodo viene chiamato sia quando si avvia la produzione, sia quando l'utente decide di cambiare il brano da riprodurre.

All'interno del metodo [eseguiBrano:] vengono inizializzati e creati tutti i nodi della catena di gestione del flusso audio di AVAudioEngine. All'interno di questo metodo vengono anche inizializzate le variabili che servono per la barra di riproduzione.

5.7.7 Implementazione della barra di riproduzione

La barra di riproduzione musicale, vale a dire la parte del codice che si occupa di mostrare qual è il brano in esecuzione e di fornire all'utente gli strumenti per controllare la riproduzione audio, è stata implementata all'interno della classe ViewController.

Graficamente la barra di riproduzione è stata creata all'interno del file Main.storyboard con elementi della libreria UIKit.

La barra di riproduzione è costituita da diversi elementi che vengono elencati nella seguente tabella.

IBOutlet	Classe	Descrizione
copertinaBranoCorrente	UIImageView	Mostra la copertina del brano in esecuzione
etichettaNomeBrano	UILabel	Mostra il titolo del brano in esecuzione
pulsantePlay	UIButton	Pulsante per play e pausa
-	UIButton	Pulsante per passaggio al brano successivo
-	UIButton	Pulsante per passaggio al brano precedente
sliderPosizione	UISlider	Mostra lo stato di riproduzione del brano. Consente di muoversi all'interno del brano.
tempoTrascorso	UILabel	Segnala quanti minuti e secondi sono trascorsi dall'inizio del brano
tempoRimanente	UILabel	Segnala quanti minuti e secondi mancano alla fine del brano

I metodi usati dalla barra di riproduzione sono elencati e descritti nella seguente tabella.

Metodo	Descrizione
[eseguiBrano:]	Usato per avviare la riproduzione di un brano
[aggiornaTempoBrano]	Metodo chiamato ciclicamente per aggiornare il valore degli IBOutlet sliderPosizione, tempoTrascorso, tempoRimanente
[pulsantePlayPremuto:]	Usato per avviare o mettere in pausa la riproduzione musicale
[pulsanteAvantiPremuto:]	Usato per passare al prossimo brano della playlist
[pulsanteIndietroPremuto:]	Usato per passare al precedente brano della playlist oppure per tornare all'inizio del brano
[sliderPosizioneModificato:]	Usato per consentire all'utente di cambiare il punto del brano in riproduzione

5.7.8 Implementazione delle playlist

La gestione della playlist, vale a dire la parte del codice che si occupa di consentire all'utente di vedere e scegliere quali brani saranno riprodotti, è stata implementata all'interno della classe ViewController.

Graficamente la barra della playlist è stata creata all'interno del file Main.storyboard con elementi della libreria UIKit.

Il menù della playlist è costituito da diversi elementi che vengono elencati nella seguente tabella.

IBOutlet	Classe	Descrizione
tabellaPlaylist	UITableView	Mostra l'elenco dei brani all'interno della playlist
switchPlaylist	UISwitch	Consente di cambiare il tipo di playlist
-	UIButton	Consente di chiudere il menù della playlist
-	UIButton	Consente di eliminare dalla memoria il grafo e la playlist

Si è scelto di implementare due tipi diversi di playlist.

Il primo tipo di playlist, chiamato "Album", inserisce automaticamente tutto l'album che contiene il brano selezionato all'interno del grafo dall'utente nella playlist. In questo modo l'utente pur avendo selezionato un solo brano è in grado di ascoltare, potenzialmente, oltre un'ora di musica

(vale a dire tutto l'album). Ogni volta che l'utente seleziona un nuovo brano dal grafo, tutta la playlist viene svuotata e ne viene creata una nuova con i brani del nuovo album.

Il secondo tipo di playlist, chiamato "Playlist", inserisce nella playlist solo il brano selezionato dall'utente. In questo modo l'utente è in grado di inserire nella playlist brani appartenenti ad album diversi.

Dopo aver creato una playlist, l'utente è in grado di eliminare dalla playlist i brani che non desidera ascoltare.

I metodi usati per la gestione della playlist sono elencati e descritti nella seguente tabella.

Metodo	Descrizione
[creaPlaylistAlbum:]	Usato per creare la playlist in modalità "album"
[aggiungiBranoAPlaylist:]	Usato per aggiungere un brano alla playlist in modalità "playlist"
[tableView: commitEditingStyle: forRowAtIndexPath:]	Usato per eliminare un brano dalla playlist

5.7.9 Implementazione della barra di ricerca

La barra di ricerca, vale a dire la parte del codice che si occupa di consentire all'utente di scegliere da quale artista, album oppure brano iniziare il grafo, è stata implementata all'interno della classe ViewController.

Graficamente la barra della playlist è stata creata direttamente nel file ViewController.m con elementi della libreria UIKit.

La barra di ricerca viene usata sia per la selezione iniziale dell'elemento dal quale costruire il grafo, sia per inserire, successivamente, altri nodi al grafo.

La barra di ricerca viene creata all'interno del metodo [creaBarraDiRicerca]. La barra di ricerca può essere estesa (allungata) o ridotta a discrezione dell'utente. La barra di ricerca può essere completamente nascosta. Al suo posto compare un pulsante che consente di riaprirla.

La barra di ricerca contiene tre pulsanti che consentono di scegliere se la ricerca deve essere effettuata tra gli artisti, tra gli album oppure tra i brani.

I metodi usati dalla barra di ricerca sono elencati e descritti nella seguente tabella.

Metodo	Descrizione
[creaBarraDiRicerca]	Usato per creare da zero la barra di ricerca
[searchBar: textDidChange:]	Usato per aggiornare l'elenco delle voci visualizzate in base a quanto digitato dall'utente. Minimo due caratteri.
[tableView: didSelectRowAtIndexPath:]	Usato per avviare l'inserimento del nodo nel grafo in base a quanto selezionato dall'utente
[cercaArtista]	Usato per aggiornare l'elenco delle voci visualizzate. Effettua una ricerca all'interno degli artisti presenti nella libreria musicale.
[cercaAlbum]	Usato per aggiornare l'elenco delle voci visualizzate. Effettua una ricerca all'interno degli album presenti nella libreria musicale.
[cercaBrano]	Usato per aggiornare l'elenco delle voci visualizzate. Effettua una ricerca all'interno dei brani presenti nella libreria musicale.

5.7.10 Prestazioni

Nonostante questa app sia basata sulla grafica e si basa su un'interfaccia che si aggiorna a 60 frames al secondo, ha ottime prestazioni ed una occupazione della CPU.

Le prove di questa App sono state fatte con un iPad Air 2 Wifi + Cellular con 128 GB di memoria.

La libreria musicale dell'iPad usato per testare questa App contiene 5761 brani.

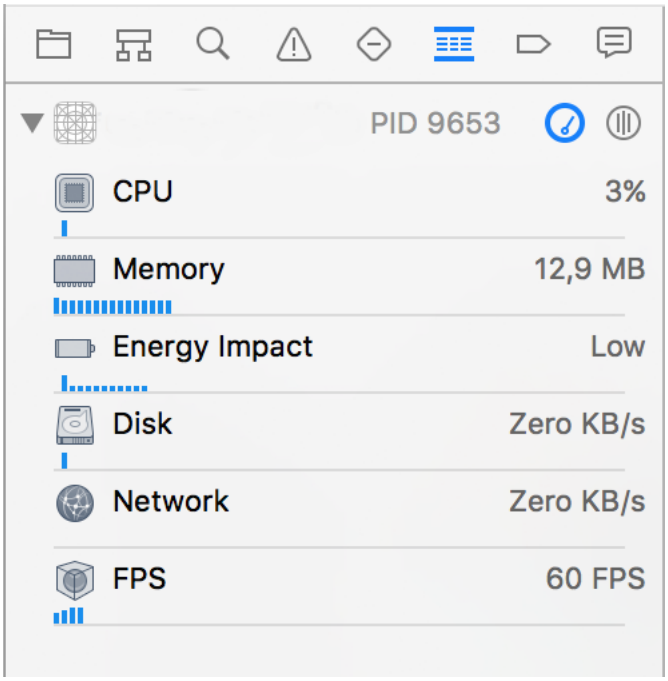


Fig. 29 Prestazioni della App all'apertura

L'immagine precedente mostra le prestazioni della App subito dopo l'apertura. Come si può vedere l'occupazione della CPU è al 3%, nonostante per sua natura l'interfaccia basata sul framework SpriteKit si aggiorni 60 volte al secondo.

Dopo aver inserito circa 70 nodi nel grafo ed aver avviato la riproduzione, l'occupazione della CPU non aumenta di molto.

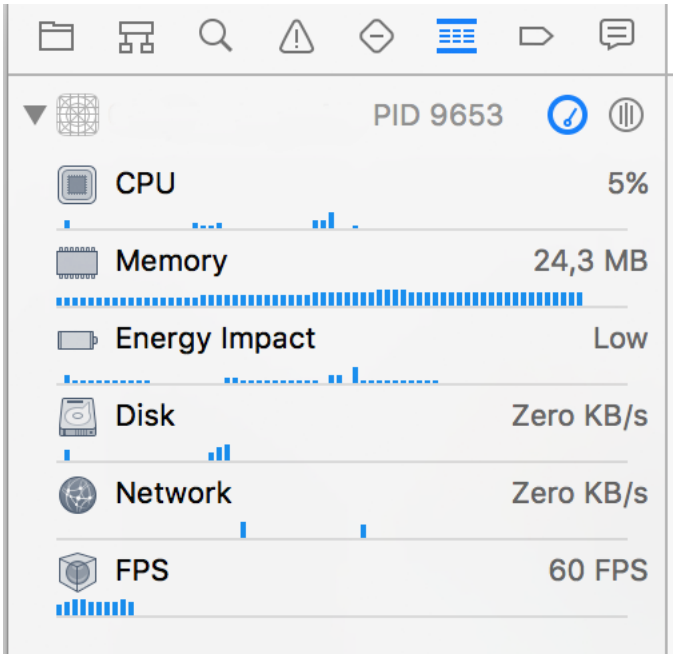


Fig. 30 Prestazioni della App dopo l'avvio della riproduzione

Come si può vedere l'occupazione della CPU è al 5%. L'impatto sul consumo di batteria è considerato "Low" da parte di Xcode.

E' possibile ridurre ulteriormente l'occupazione della CPU diminuendo il refresh rate di SpriteKit ad esempio da 60 a 30 fps.

Capitolo 6 – Layout e funzionamento del player musicale

In questo capitolo vengono analizzati gli aspetti di layout e di funzionamento della App.

6.1 Layout della App

L'interfaccia della App è stata creata usando uno stile il più pulito possibile, in conformità a quello che è lo standard per questo tipo di Applicazioni.

L'immagine seguente mostra la App subito dopo l'apertura.

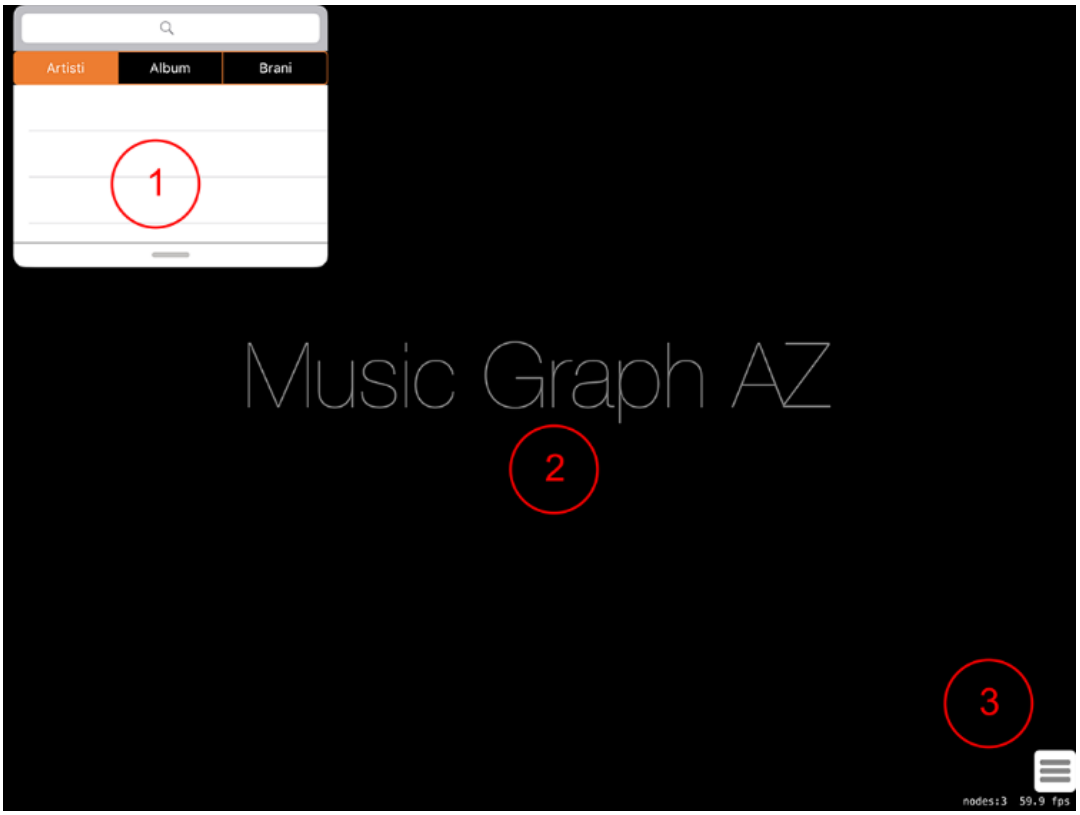


Fig. 31 App subito dopo l'apertura

Nell'immagine sono stati evidenziati gli elementi del layout immediatamente visibili. Il primo è il menù di ricerca, il secondo è l'ampio spazio dedicato alla visualizzazione del grafo, il terzo è il pulsante che consente di accedere alla playlist. Vicino al pulsante per l'accesso alla playlist è visibile un piccolo testo che mostra il numero di nodi visibile ed il numero di frame per secondo.

Dopo aver inserito un artista del grafo, il layout della App si modifica in quanto il menù di ricerca viene nascosto per dare maggior risalto al grafo.

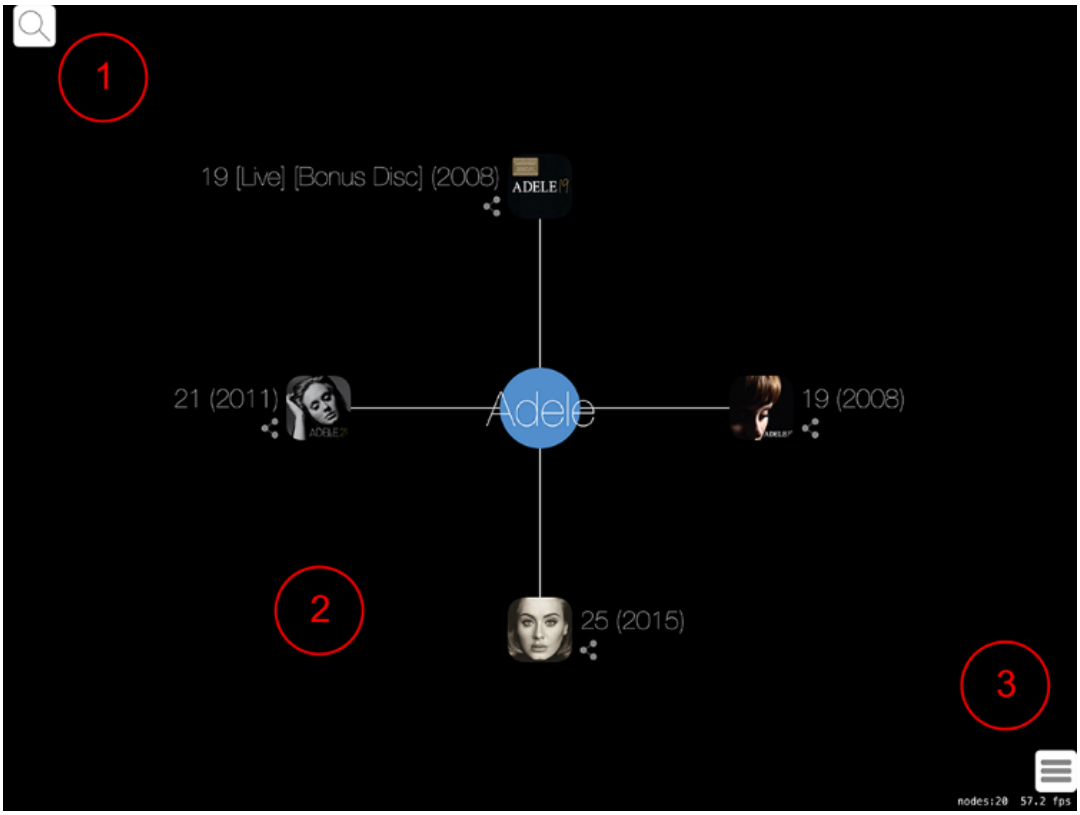


Fig. 32 App dopo l'inserimento di un artista del grafo

Nell'immagine sono stati evidenziati gli elementi del layout visibili. Il primo è l'icona per l'apertura del menù di ricerca, il secondo è l'ampio spazio dedicato alla visualizzazione del grafo, il terzo è il pulsante che consente di accedere alla playlist.

Dopo aver avviato la avviato la riproduzione musicale, facendo doppio tap sul nodo del brano, la App fa comparire un ulteriore elemento dell'interfaccia. Si tratta della barra di riproduzione che contiene tutti gli elementi necessari all'utente per poter controllare la riproduzione.

All'interno della barra di riproduzione si trovano:

- la copertina del brano
- il titolo del brano
- il pulsante play/pausa
- i pulsanti per cambiare brano in avanti o indietro
- le etichette con il tempo trascorso ed il tempo rimanente alla fine del brano

- lo slider che mostra l'avanzamento della riproduzione e che consente all'utente di modificare il punto della riproduzione all'interno del brano.

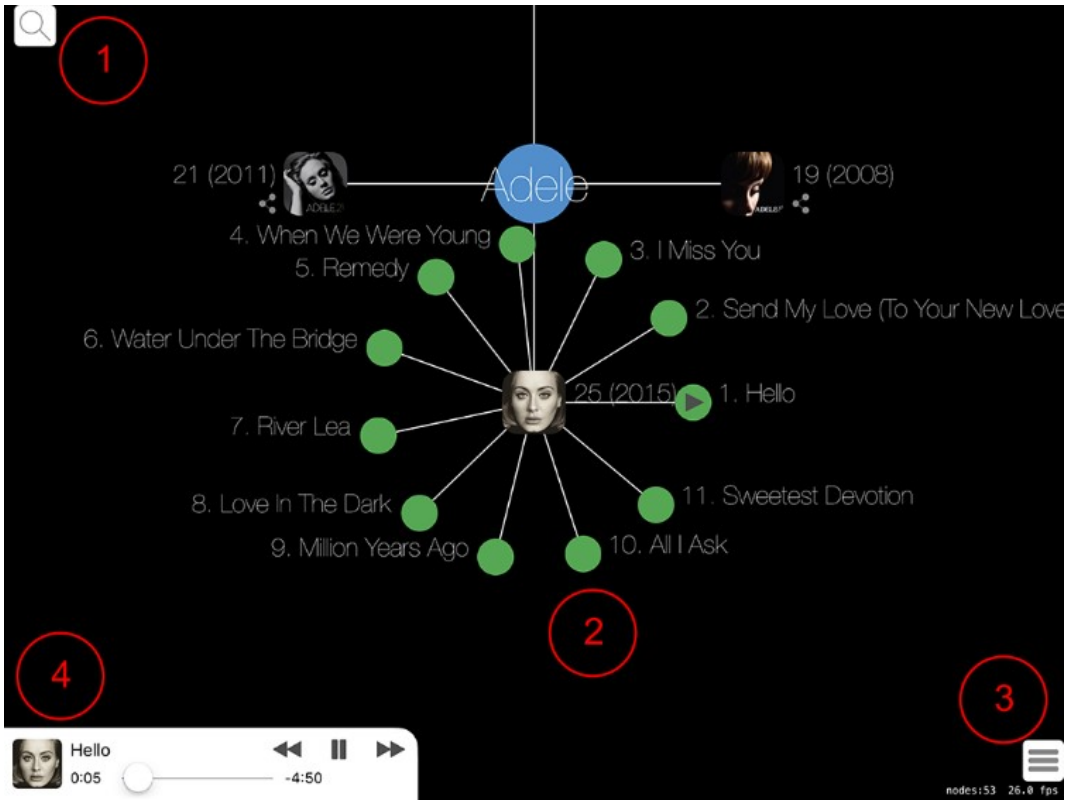


Fig. 33 App dopo aver avviato la riproduzione

Nell'immagine sono stati evidenziati gli elementi del layout visibili. Il primo è ancora l'icona per l'apertura del menù di ricerca, il secondo è l'ampio spazio dedicato alla visualizzazione del grafo, il terzo è il pulsante che consente di accedere alla playlist ed il quarto è la barra di riproduzione appena mostrata.

Esiste un ulteriore elemento dell'interfaccia che è importante analizzare. Si tratta della barra dedicata alla gestione della playlist. Questo pannello si trova sul lato destro dello schermo ed include diverse funzionalità:

- consente di vedere l'elenco dei brani contenuto nella playlist
- consente di passare tramite uno slider dalla modalità album alla modalità playlist
- consente di scegliere un brano dalla playlist
- consente di cancellare un brano dalla playlist
- consente di eliminare la playlist ed il grafo

La seguente immagine mostra la App con un brano in riproduzione ed il pannello dedicato alla playlist aperto.

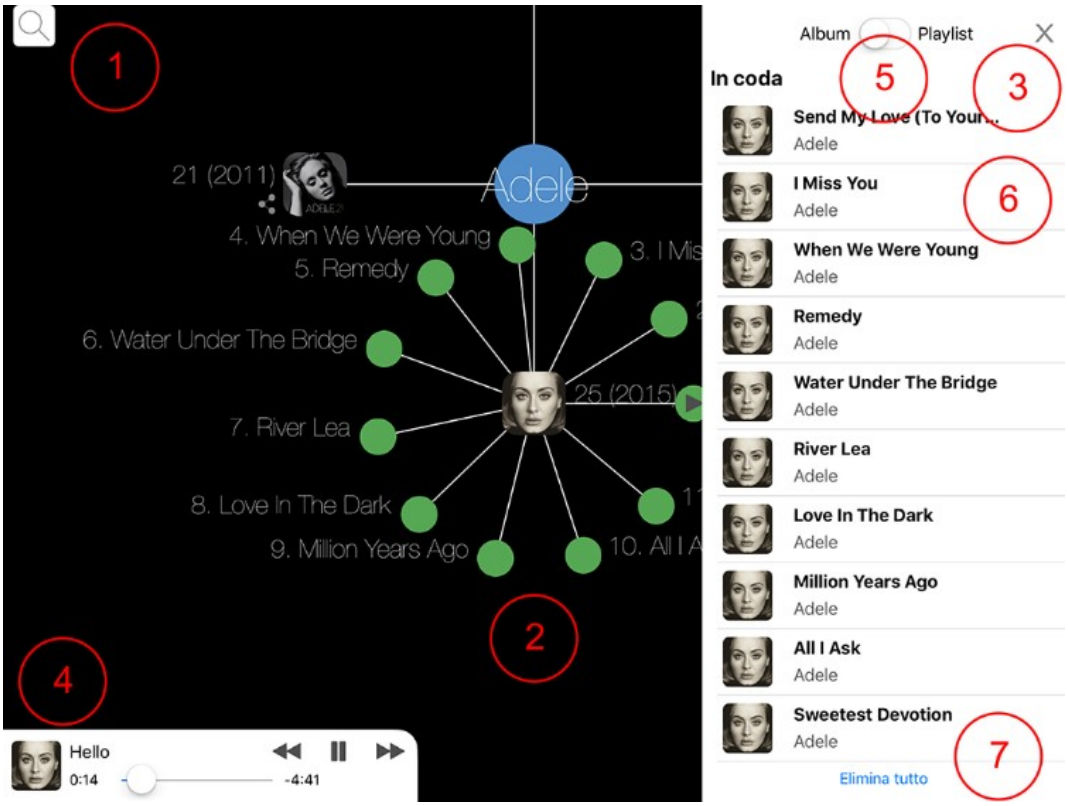


Fig. 34 App con il pannello dedicato alla playlist aperto

Nell'immagine sono stati evidenziati gli elementi del layout visibili. Il primo è ancora l'icona per l'apertura del menù di ricerca, il secondo è l'ampio spazio dedicato alla visualizzazione del grafo, il terzo è il pulsante che consente di chiudere il pannello della playlist, il quarto è la barra di riproduzione, il quinto è lo slider che consente di passare dalla modalità album alla modalità playlist, il sesto è l'elenco dei brani nella playlist ed il settimo è il pulsante per cancellare la playlist ed il grafo.

E' importante notare che l'elemento numero sei in figura svolge tre funzioni:

- mostra l'elenco dei brani già nella playlist
- consente di cancellare i singoli brani dalla playlist (swipe verso sinistra)
- consente di scegliere il brano da ascoltare (con un tap).

La funzione di cancellazione svolta dal pulsante numero sette consente all'utente di ripartire da zero senza dover chiudere e riaprire la App.

6.2 Funzionamento del player musicale

In questa sezione viene descritto il funzionamento del player musicale.

L'utilizzo di questa App è semplice, ma è comunque opportuno elencare i passaggi che servono per utilizzare la App.

Quando si inizia ad usare la App lo schermo è vuoto e viene richiesto all'utente, tramite il menu di ricerca, di selezionare da quale nodo desidera sviluppare il grafo.

L'utente può scegliere alternativamente un artista, un album oppure un brano.

A seconda della selezione effettua, il risultato sarà diverso.

Se l'utente seleziona un artista la App rappresenta sullo schermo un grafo costituito da un nodo per l'artista ed un nodo per ciascun album dall'artista contenuto nella libreria dell'iPad.

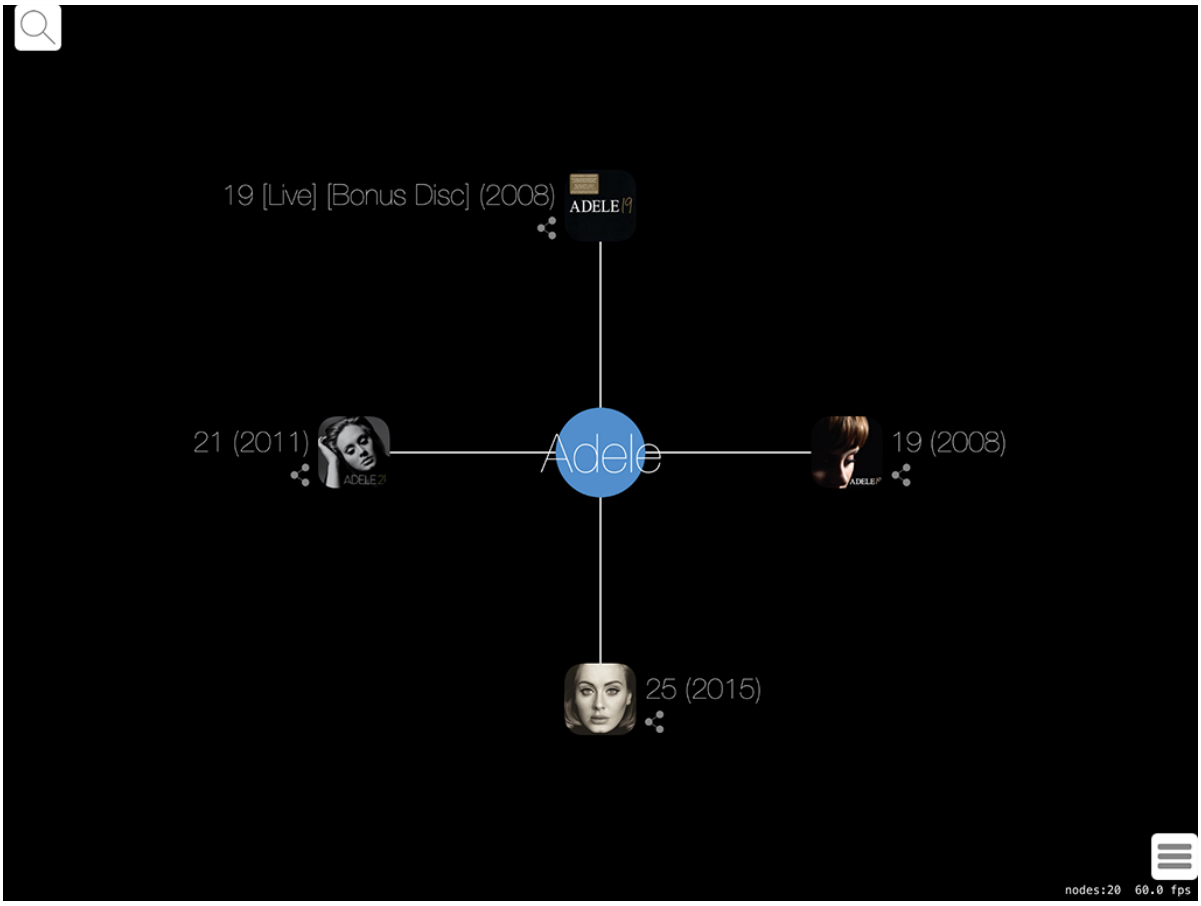


Fig. 35 L'utente ha inserito un artista

Se l'utente seleziona un album, la App rappresenta sullo schermo un grafo costituito da un nodo per l'artista, un nodo per l'album ed un nodo per ciascun brano dell'album contenuto nella libreria dell'iPad.

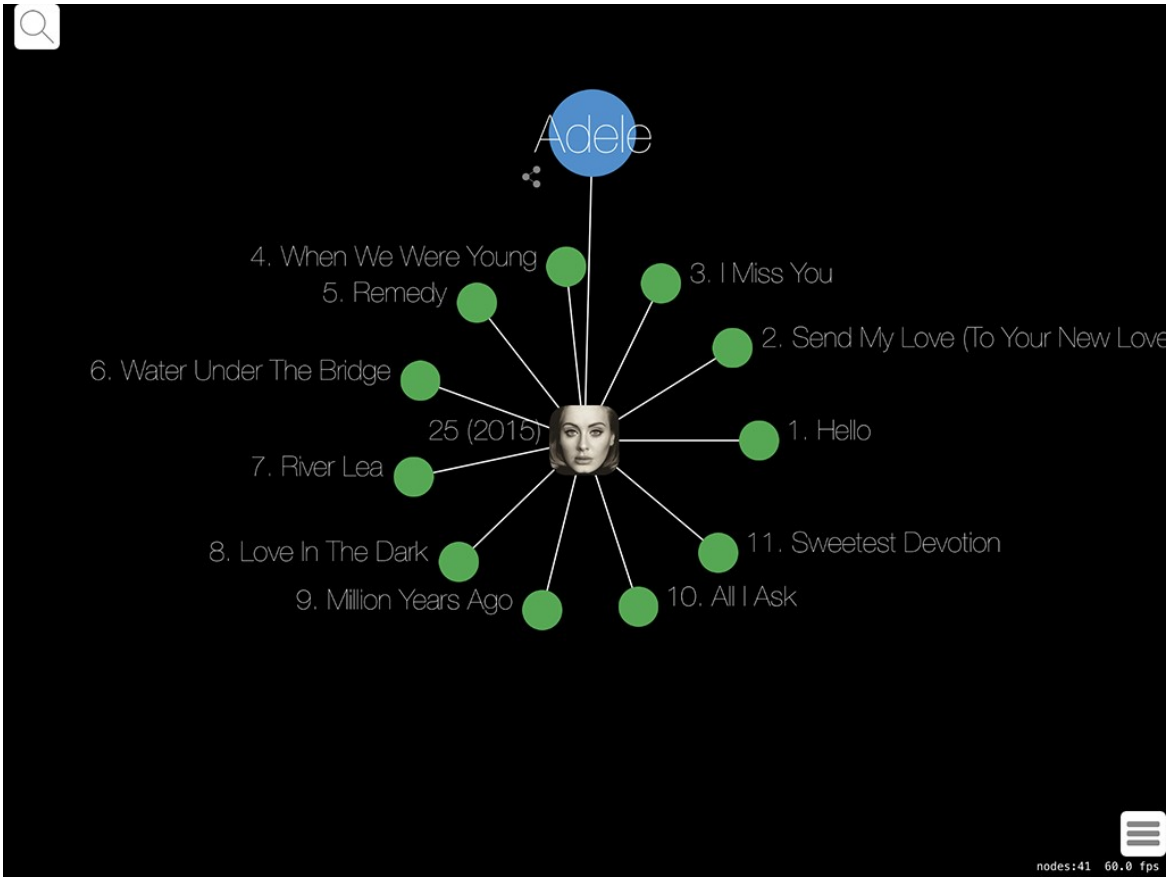


Fig. 36 L'utente ha inserito un album

Se l'utente seleziona un brano, la App rappresenta sullo schermo un grafo costituito da un nodo per l'artista, un nodo per l'album ed un nodo per il brano.

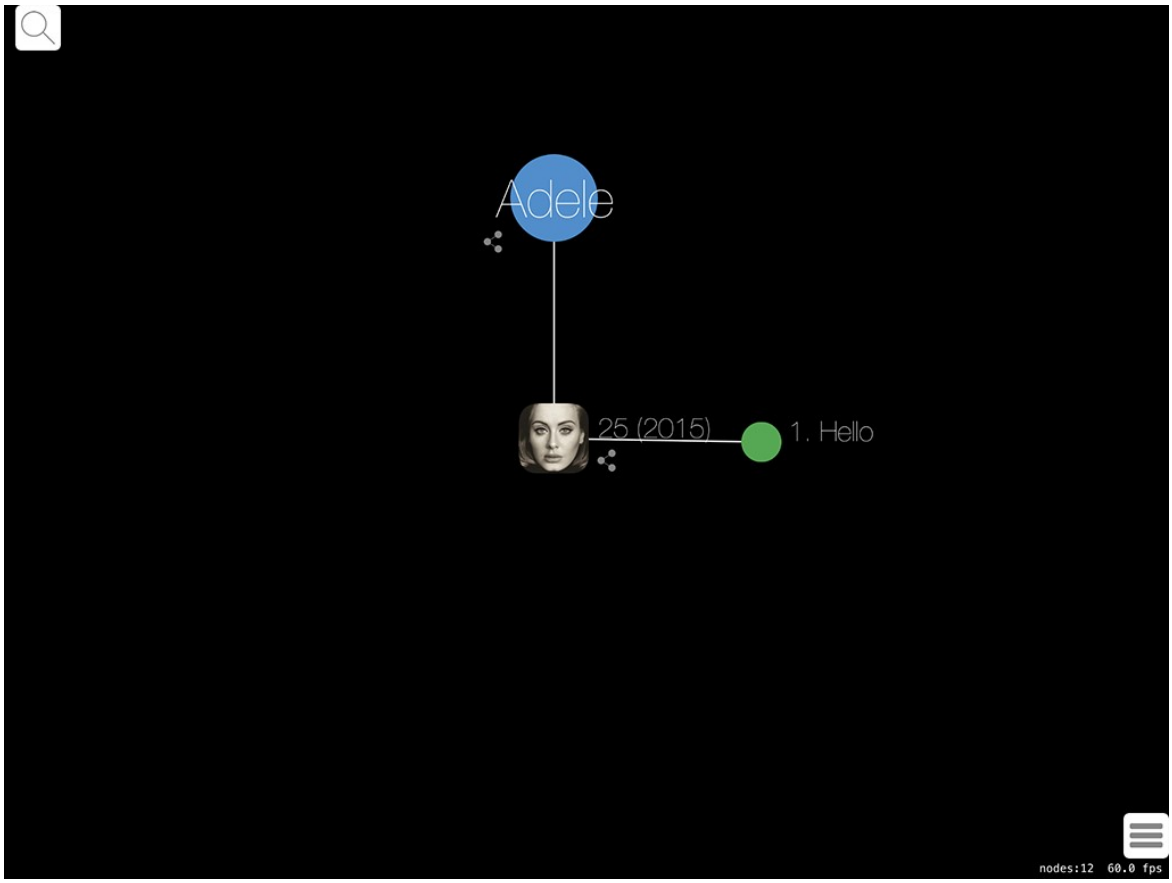


Fig. 37 L'utente ha inserito un brano

Un apposito simbolo grafico sul nodo segnala la possibilità di espandere il grafo.

Fig. 38 Simbolo che indica la presenza di collegamenti disponibili

Quando questo simbolo è posizionato vicino ad un album, toccando l'album vengono mostrati i brani dell'album. In questo modo l'utente può esplorare tutti gli album ed i brani dell'artista presenti nella libreria.

Quando questo simbolo è posizionato vicino ad un artista, toccando il nodo dell'artista si ottengono due possibili risultati. Se gli album dell'artista non sono ancora visibili sullo schermo, la prima volta che si tocca il nodo dell'artista vengono mostrati tutti i suoi album presenti nella libreria musicale dell'iPad. Se il simbolo dei collegamenti è poi ancora visibile, significa che questo artista ha un collegamento avanzato con un altro artista. Questo tipo di collegamenti si basa sulla banca dati online Discogs.

L'utente può scegliere di inserire nel grafo un altro artista (oppure album o brano) e quindi creare un grafo non connesso.

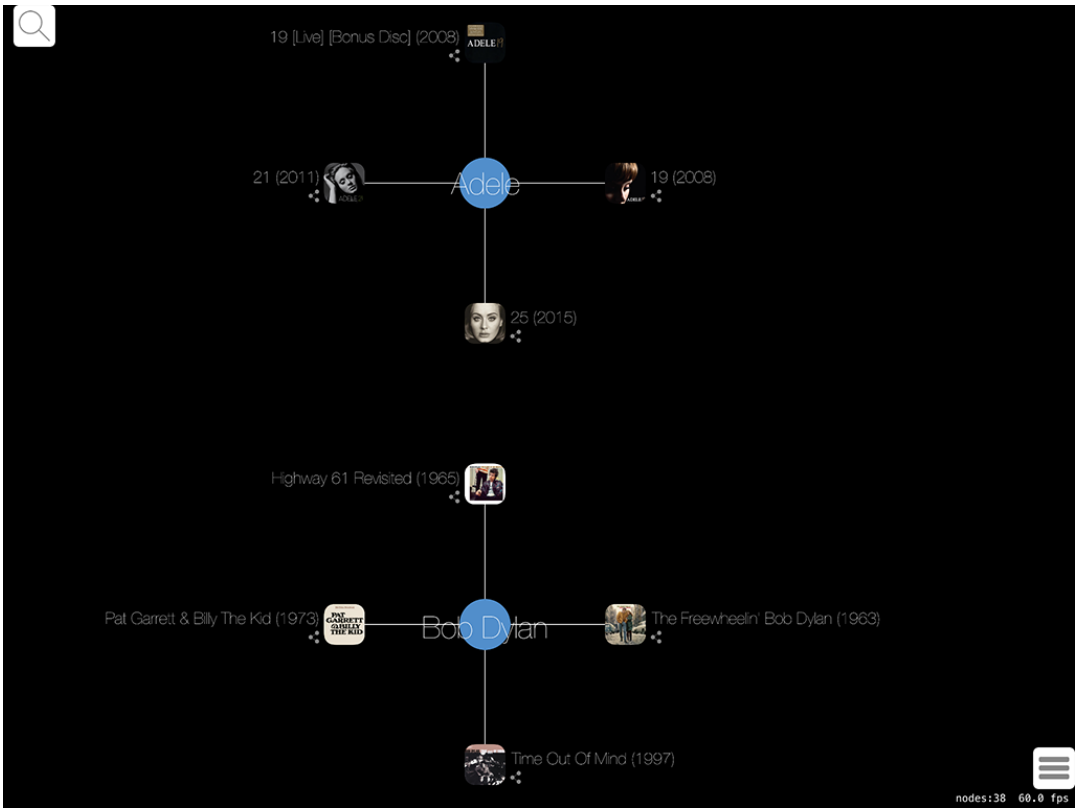


Fig. 39 Esempio di grafo non connesso

L'esplorazione del grafo può portare il grafo a tornare ad essere connesso.

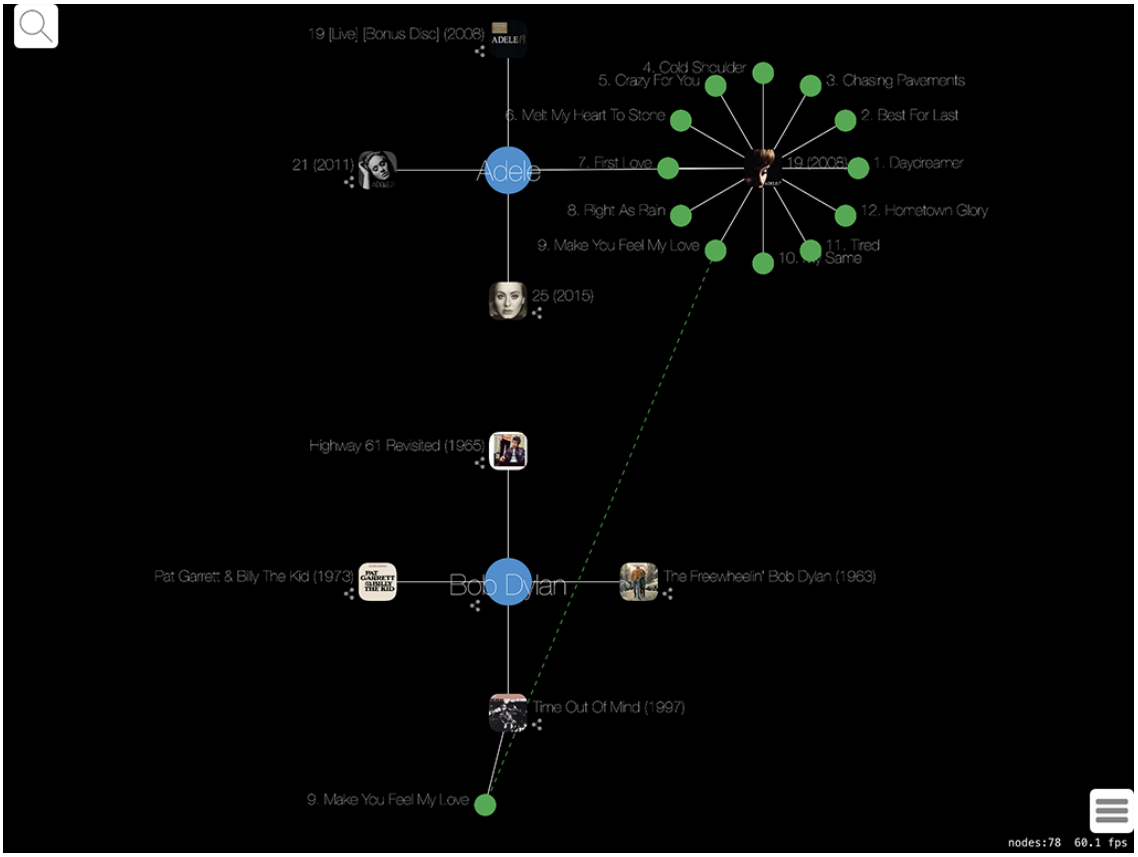


Fig. 40 Grafo connesso

Nell'esempio precedente due brani di due album e due artisti differenti contengono una cover, pertanto la App crea un collegamento (verde e tratteggiato) che rende il grafo connesso.

La riproduzione audio di un brano è avviata tramite il nodo corrispondente al brano. Dopo aver fatto doppio tap parte la riproduzione ed il nodo del brano che si sta ascoltando mostra un simbolo che segnala il suo stato di brano in esecuzione.

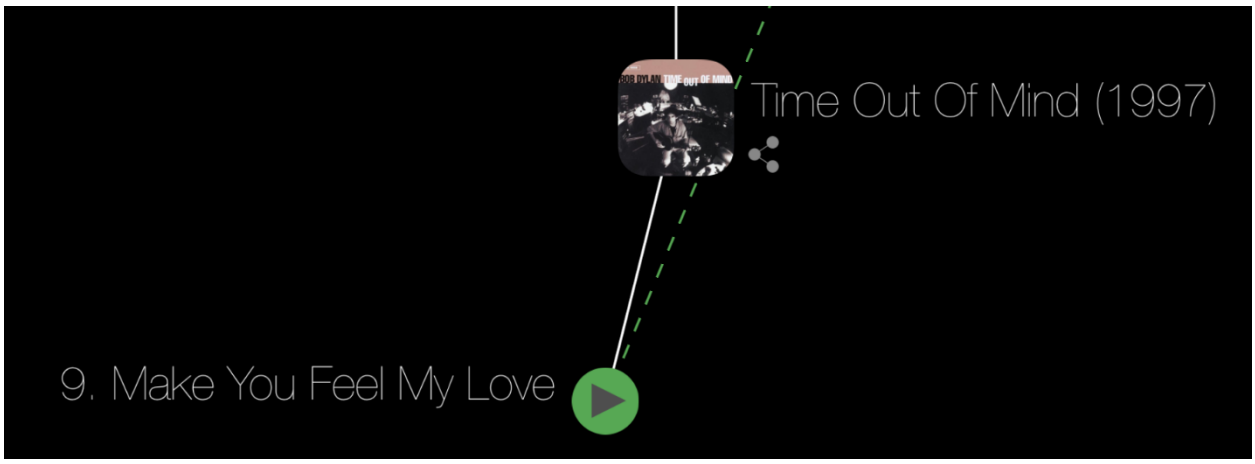


Fig. 41 Nodo del brano in esecuzione

La App segnala se lo stesso brano è presente in più album (ad esempio raccolte di brani non nuovi). Gli archi che collegano cover e brani identici tra loro sono rappresentati diversamente dagli altri archi per una facile identificazione. In questo caso viene usata una linea verde continua.

Come già anticipato, la App è in grado di creare collegamenti tra un artista ed altri artisti o gruppi musicali presenti nella libreria, sulla base di relazioni di natura artistica derivate in modo dinamico dalla banca dati online Discogs. La App è in grado di collegare un artista che abbia iniziato la propria carriera in un gruppo musicale e che poi abbia intrapreso la carriera da solista.

Nell'esempio mostrato nella seguente figura, l'utente è partito selezionando l'artista "John Lennon". La App è stata in grado di trovare il collegamento tra questo artista e l'artista (gruppo) "The Beatles" presente nella libreria dell'iPad. L'immagine rappresentata si riferisce alla situazione in cui l'utente ha già cliccato sul nodo del primo artista per poter visualizzare gli altri nodi collegati.

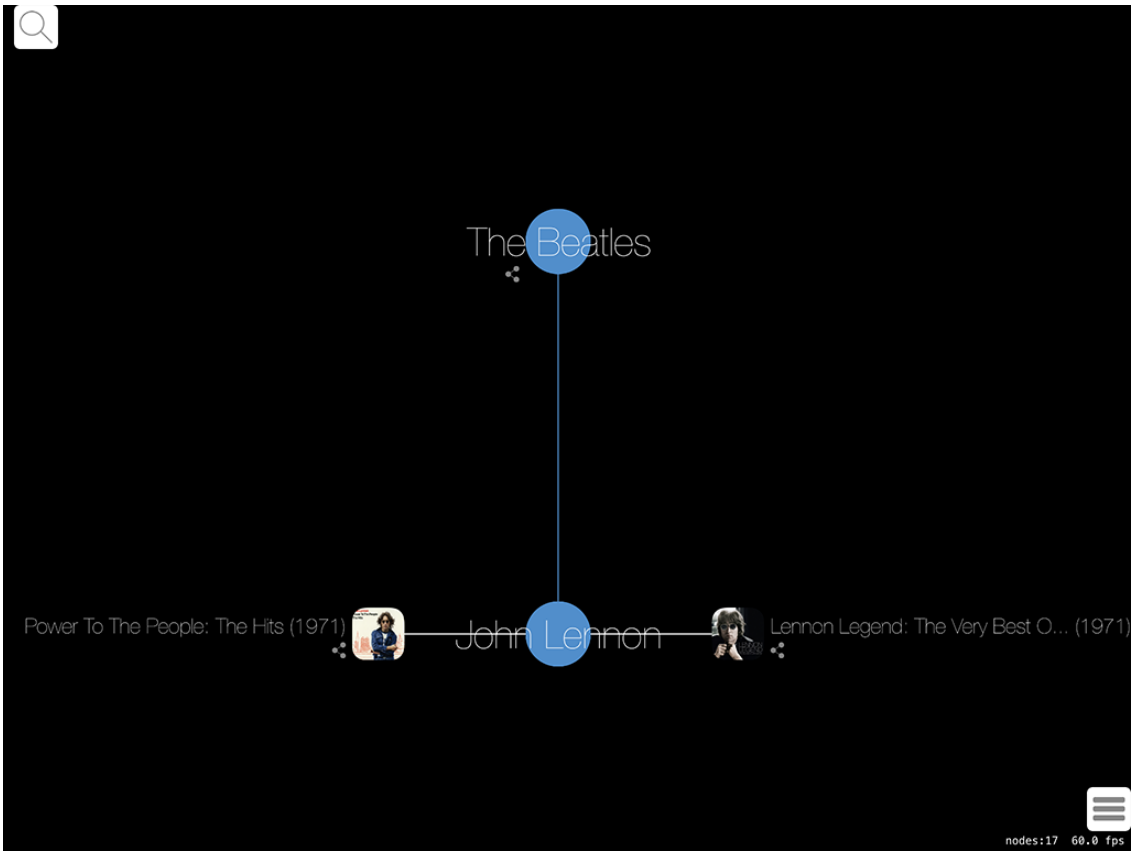


Fig. 42 Due artisti collegati

In conclusione, con la prossima figura si mette in evidenza che la rappresentazione tramite grafo della libreria musicale dell'iPad consente, ad esempio, di mostrare contemporaneamente più album. Questa funzionalità non è presente nei più diffusi player musicali disponibili sul mercato.

La possibilità di visualizzare, ad esempio, due album contemporaneamente consente diverse nuove possibilità di fruizione della musica. Ad esempio l'utente può ascoltare e confrontare brani contenuti in album differenti oppure può creare una playlist selezionando brani da album diversi ma senza la necessità di passare da un album all'altro.

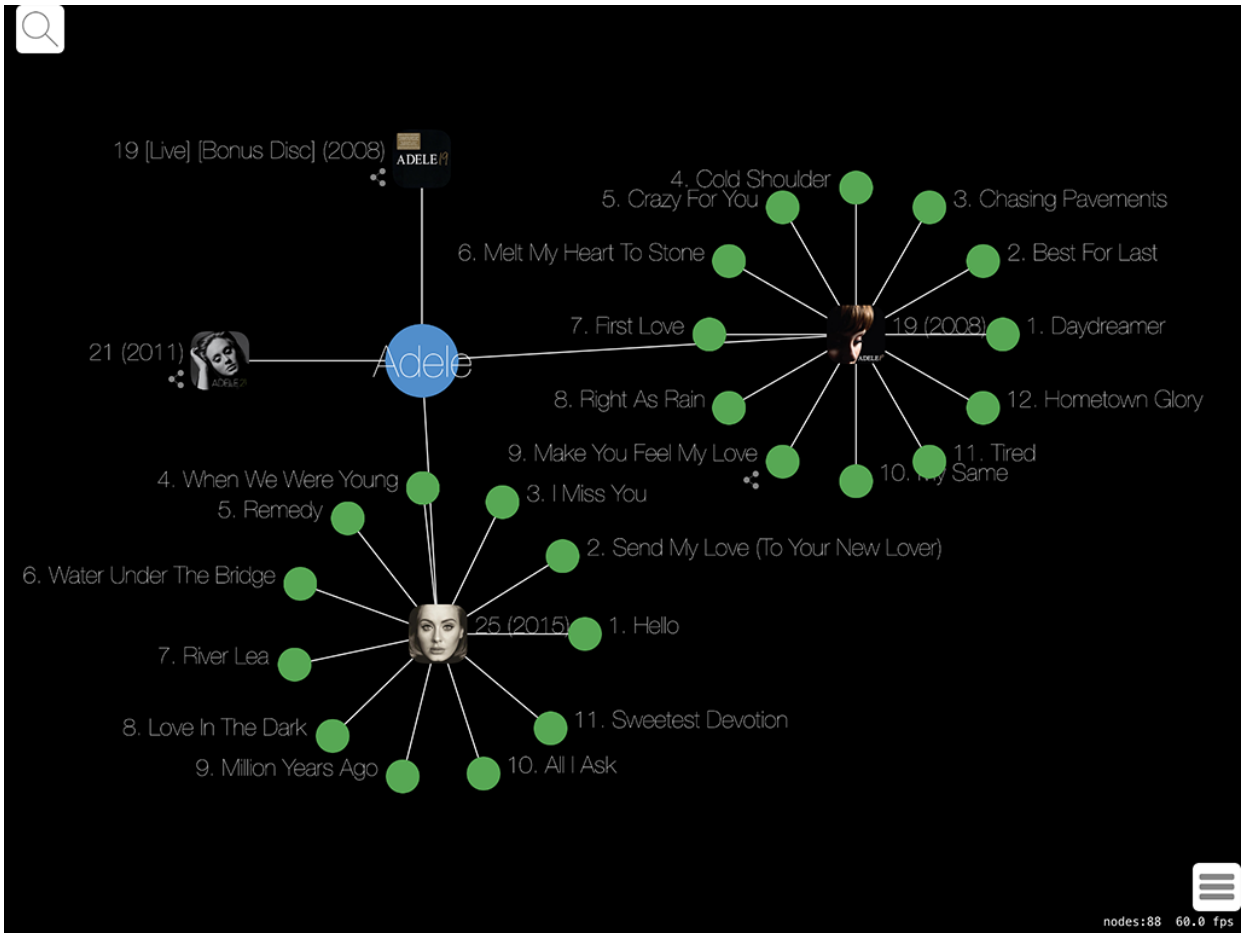


Fig. 43 Due album aperti contemporaneamente

Questo tipo di visualizzazione affiancata è possibile non solo con gli album ma anche con gli artisti. E' inoltre possibile affiancare anche singoli brani presi da album differenti.

Capitolo 7 - Conclusioni e sviluppi futuri

In questo capitolo vengono analizzate le conclusioni di questo lavoro di tesi e vengono proposti alcuni sviluppi possibili per l'evoluzione di quanto è stato realizzato in questa tesi.

7.1 Conclusioni

Questa tesi è nata come risposta ai limiti dell'interfaccia utente dei player musicali diffusa nei dispositivi in commercio. Tutte le interfacce utente attualmente più diffuse si basano su una rappresentazione tabellare della libreria musicale.

La rappresentazione in tabella della libreria musicale sebbene molto semplice e intuitiva, a volte può rendere macchinosa la ricerca all'interno di banche dati musicali molto ricche, specialmente sui piccoli schermi dei device mobili. Inoltre, la rappresentazione in tabella non consente di mettere facilmente in relazione brani appartenenti a diversi artisti e a diversi album.

Lo scopo di questo lavoro di tesi era quello di dimostrare che è possibile rappresentare le librerie musicali usando forme alternative alla rappresentazione tabellare. In particolare si è scelto di rappresentare la libreria musicale usando la teoria dei grafi.

Per dimostrare che la tesi era corretta, si è deciso di realizzare una App per la riproduzione musicale, sviluppata per tablet iPad di Apple completamente basata sulla teoria dei grafi. La App che è stata realizzata per questa tesi permette di navigare all'interno della libreria musicale tramite un grafo che collega tra loro i brani, gli album e gli artisti, adottando criteri diversi.

La scelta di basare la App sulla teoria dei grafi si è rivelata corretta ed ha permesso di realizzare una soluzione intuitiva, versatile, di semplice utilizzo ed efficace nel consentire all'utente navigazioni rapide all'interno di librerie con migliaia di brani.

Gli elementi che vengono rappresentati nel grafo sono gli artisti, gli album ed i brani. La nuova interfaccia grafica consente di effettuare ricerche e di fare collegamenti per meglio apprezzare i brani e per scoprire e riscoprire legami e collegamenti tra artisti, album e brani diversi.

La App che è stata realizzata offre diverse funzionalità e consente all'utente di selezionare da quale nodo desidera sviluppare il grafo. L'utente può scegliere alternativamente un artista, un album oppure un brano. Se l'utente seleziona, ad esempio, un artista la App rappresenta sullo schermo un grafo costituito da un nodo per l'artista ed un nodo per ciascun album dall'artista contenuto nella libreria dell'iPad.

Sullo schermo un apposito simbolo grafico sul nodo di ogni album segnala la possibilità di espandere il grafo, toccando l'album, in modo che siano mostrati i brani dell'album. In questo modo l'utente può esplorare tutti gli album ed i brani dell'artista presenti nella libreria.

L'utente può anche scegliere di inserire nel grafo un altro artista (oppure album o brano) e quindi creare un grafo non connesso. L'utente può avviare la riproduzione audio di un brano tramite il nodo del grafo corrispondente al brano.

La App che è stata realizzata è in grado di creare collegamenti tra brani presenti nel grafo. Ad esempio la App crea un collegamento tra cover ovvero reinterpretazioni di brani musicali da parte di qualcuno che non ne è l'interprete originale. Inoltre la App segnala se lo stesso brano è presente in più album (ad esempio raccolte di brani non nuovi). Gli archi che collegano cover e brani identici tra loro sono rappresentati diversamente dagli altri archi per una facile identificazione;

La App è inoltre in grado di creare collegamenti tra un artista ed altri artisti o gruppi musicali presenti nella libreria, sulla base di relazioni di natura artistica derivate in modo dinamico da una banca dati online. Ad esempio è in grado di collegare un artista che abbia iniziato la propria carriera in un gruppo musicale e che poi abbia intrapreso la carriera da solista.

Lo sviluppo di questa App ha richiesto la realizzazione di una libreria appositamente scritta per la rappresentazione dei grafi, dato che allo stato attuale non esistono librerie disponibili con tali funzionalità.

La scelta di basare la App sulla teoria dei grafi si è rivelata corretta in quanto ha consentito non solo di rappresentare la libreria musicale, ma anche di inserire nuovi tipi di collegamenti tra brani e artisti che non sono presenti nei player con rappresentazione tabellare. Inoltre la rappresentazione tramite grafo ha consentito di mostrare sullo schermo più album contemporaneamente, consentendo all'utente di visualizzare allo stesso tempo più album con la possibilità quindi di creare più semplicemente la propria playlist.

In conclusione, questo lavoro di tesi dimostra che la fruizione delle librerie musicali tramite grafi non solo è possibile ma è anche intuitiva e di semplice usabilità, alternativa all'impiego di

tabelle. La rappresentazione tramite grafi offre nuovi ed innovativi modi di fruizione della libreria musicale stessa

7.2 Sviluppi futuri

Questo lavoro di tesi ha aperto la strada per un nuovo tipo di player musicale dove la rappresentazione della libreria musicale avviene tramite grafo e non tramite tabelle.

Per gli sviluppi futuri è possibile pensare a nuove forme di collegamento tra i nodi del grafo basate su:

- analisi dei metadati presenti all'interno della libreria musicale del device;
- analisi dei metadati presenti nelle banche dati online;
- analisi audio dei brani.

I primi due tipi di analisi sono stati in parte già analizzati in questo lavoro di tesi. L'analisi audio dei brani potrebbe consentire un ulteriore espansione del numero e della qualità dei collegamenti tra i nodi del grafo.

Bibliografia

[1] Apple Inc., «Apple Quicktime,» [Online]. Available: http://www.apple.com/it/quicktime/.

[2] Microsoft Corporation, «Microsoft Windows Media Player,» [Online]. Available: http://windows.microsoft.com/it-it/windows/windows-media-player.

[3] Fraunhofer Institute, «Fraunhofer Institute,» [Online]. Available: http://www.fraunhofer.de.

[4] Winamp, «Winamp,» [Online]. Available: http://www.winamp.com/.

[5] Apple Inc., «Apple iTunes,» [Online]. Available: http://www.apple.com/it/itunes/.

[6] Apple Inc., «Apple iPhone,» [Online]. Available: http://www.apple.com/it/iphone/.

[7] Apple Inc., «Apple iPad,» [Online]. Available: http://www.apple.com/it/ipad/.

[8] A. Andric, P.-L. Xech e A. Fantasia, «Music Mood Wheel: Improving browsing experience on digital content through an audio interface,» in *Proc. of the 2nd International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution (AXMEDIS'06)*, Leeds, UK, 2006.

[9] A. Andric e H. Goffredo, «Automatic playlist generation based on tracking user's listening habits,» *Multimedia Tools and Applications,* vol. 29, n. 2, pp. 127-151, June 2006.

[10] A. Andric e G. Haus, «Estimating Quality of Playlists by Sight,» in *Proc. of the First International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution (AXMEDIS'05)*, Florence, Italy, 2005.

[11] S. Pauws e B. Eggen, «PATS: Realization and User Evaluation of an Automatic Playlist Generator,» in *Proc. of the 3rd International Conference on Music Information Retrieval (ISMIR 2002)*, Paris, France, 2002.

[12] H. F. Abeles, «Responses to music,» in *Handbook of music psychology*, Lawrence, KS: National Association for Music Therapy, 1980, pp. 105-140.

[13] IDC International Data Corporation, «Worldwide Tablet plus 2-in-1 Shipments, Market Share, …,» 12 03 2015. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS25480015.

[14] StatCounter, «StatCounter Top 7 Tablet OSs,» [Online]. Available: http://gs.statcounter.com/#tablet-os-ww-monthly-201409-201509.

[15] Apple Inc., «Apple iOS,» [Online]. Available: https://developer.apple.com/ios/.

[16] A. Picchi, Programmare con Objective-C 2.0 per iOS e OS X - 2° ediz., Edizioni LSWR, 2015.

[17] Apple Inc., «App Store Developer Support,» [Online]. Available: https://developer.apple.com/support/app-store/.

[18] Apple Inc., «Apple Xcode,» [Online]. Available: https://developer.apple.com/xcode/.

[19] K. Lee, Pro Objective-C, Apress, 2013, p. Cap.1.

[20] A. Bertoni e M. Goldwurm, Progetto e analisi di algoritmi, Milano: Rapporto interno n. 230-98, Dip. di Informatica, Università degli Studi di Milano, 2011.

[21] B. Bollobás, Modern Graph Theory, Springer, 1998.

[22] R. Diestel, Graph Theory, Springer, 2010.

[23] J. Hoffman, H.-E. Grönlund, C. Francis e S. Grimes, iOS 7 Development Recipes: A Problem-Solution Approach, Apress, 2013.

[24] Apple, «iPod Library Access Programming Guide,» [Online]. Available: https://developer.apple.com/library/content/documentation/Audio/Conceptual/iPodLibraryAccess_Guide/Introduction/Introduction.html.

[25] Apple, «Delivering an Exceptional Audio Experience (Session 507),» in *WWDC 2016*, San Francisco, 2016.

[26] A. Bakir, Beginning iOS Media App Development, Apress, 2014.

[27] Apple, «AudioSession and MultiRoute Audio (Session 505),» in *WWDC 2012*, San Francisco, 2012.

[28] Apple, «AVAudioEngine in Practice (Session 502),» in *WWDC 2014*, San Francisco, 2014.

[29] Apple, «What's New in Core Audio (Session 507),» in *WWDC 2015*, San Francisco, 2015.

[30] «Gracenote,» [Online]. Available: http://www.gracenote.com/.

[31] «AllMusic,» [Online]. Available: http://www.allmusic.com/.

[32] «Discogs,» [Online]. Available: https://www.discogs.com.

[33] «Discogs API,» [Online]. Available: https://www.discogs.com/developers/.

[34] Apple, «What's New in SpriteKit,» in *WWDC 2016*, San Francisco, 2016.

[35] Apple, «SpriteKit Programming Guide,» [Online]. Available: https://developer.apple.com/library/content/documentation/GraphicsAnimation/Conceptual/SpriteKit_PG/Introduction/Introduction.html.

[36] Apple, «GameplayKit Programming Guide,» [Online]. Available: https://developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/index.html.

[37] Apple, «Introduction to GameplayKit,» in *WWDC 2015*, San Francisco, 2015.

Ringraziamenti

Desidero ricordare tutti coloro che mi hanno aiutato nella stesura della tesi con suggerimenti, critiche ed osservazioni: a loro va la mia gratitudine.

Ringrazio anzitutto il Dott. Luca Andrea Ludovico, Relatore, ed il Dott. Adriano Baratè, Co-relatore: il loro supporto è stato prezioso per poter realizzare questa tesi.

Ringrazio, inoltre, il personale del LIM, Laboratorio di Informatica Musicale del Dipartimento di Informatica, per avermi accolto in questo prestigioso laboratorio.