FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Spotify-ed: Music Recommendation and Discovery in Spotify

**José Lage Bateira**

PREPARAÇÃO DA DISSERTAÇÃO


Mestrado Integrado em Engenharia Informática e Computação

Orientador: Fabien Gouyon

31 de Janeiro de 2014

# Spotify-ed: Music Recommendation and Discovery in Spotify

**José Lage Bateira**

Mestrado Integrado em Engenharia Informática e Computação

31 de Janeiro de 2014

# Resumo

# Abstract

# Conteúdo

# Lista de Figuras

LISTA DE FIGURAS

# Lista de Tabelas

LISTA DE TABELAS

# Abreviaturas e Símbolos

API Aplication Programming Interface

# Capítulo 1

# Introdução

## 1.1 Contexto e Enquadramento

Bem longe vão os tempos, antes da Internet, em que ouvir e descobrir música nova era um desafio por si só. Agora, com alguns cliques, temos acesso a um catálogo de música tão grande, que o nosso cérebro não consegue processar.

Existem dezenas de serviços online que oferecem isso mesmo. Alguns especializam-se na criação/geração de playlists (que funcionam como rádios), outros em expandir o catálogo de música e outros focam-se mais na sugestão e recomendação de artistas/álbuns/músicas personalizada para os utilizadores. Estes últimos, apresentam as sugestões de conteúdo ao utilizador de uma forma rudimentar como listas ou em grelha.

No entanto, listas ou grelhas não fornecem ao utilizador qualquer tipo de informação adicional sobre a relação entre os artistas nem justificam a sua semelhança [1]. Até fazem parecer que não existe nenhuma relação/ligação entre os artistas recomendados, o que não é verdade.

Essas relações existem e podem ser representadas como uma rede de artistas interligados num grafo, onde cada nó é um artista de música, e cada ligação entre nós representa uma ligação forte de parecença entre os artistas. Este é o conceito que o RAMA[1], projeto desenvolvido no INESC Porto[2], usa. [2] [3] [4] [5]

## 1.2 Motivação e Objetivos

A partir de uma pesquisa de um artista de música, o RAMA cria e desenha um grafo que ajuda o utilizador a explorar música que lhe possa interessar de uma forma muito mais natural e informativa. A informação que o RAMA usa é retirada do serviço Last.fm[3].

[1] http://rama.inescporto.pt
[2] http://inescporto.pt
[3] http://last.fm

No entanto, quando um utilizador pretende ouvir uma música de um artista, é usado *stream* do Youtube[4]. Apesar de este oferecer um catálogo alargado de música, o mesmo não é indicado para esta funcionalidade pois não fornece uma API[5] nativamente orientada a música, nem a qualidade de som do *stream* é adequada.

A experiência musical do utilizador do RAMA poderá melhorar consideravelmente ao colmatar esta falha. Existe por isso uma necessidade de substituir o Youtube por outro serviço mais orientado a *streaming* de música de qualidade. O Spotify[6] é um deles. Fornece API orientada a música [7], e o *streaming* é de qualidade adequada para este tipo de funcionalidade.

De que formas é que se pode integrar o RAMA e o Spotify?

Por forma a resolver este problema, foram analisadas várias possibilidades:

**Spotify Play Button**[8]

*Widget* do Spotify que pode ser embebida no RAMA.

**Integrar o Perfil de um utilizador Spotify no RAMA**

Para complementar as recomendações de artistas.

**Aplicação Spotify**[9]

Serve como *plugin* ao programa do Spotify, estendendo as funcionalidades do Spotify com visualização gráfica de recomendações e construção de playlists.

**Aplicação Móvel**

Com as funcionalidades acima descritas.

A escolha final foi desenvolver uma aplicação (como *plugin*) para o Spotify. Será que um utilizador Spotify ao descobrir música nova de uma forma mais gráfica terá uma experiência de utilizador mais rica e natural do que o modo de descoberta *standard* do Spotify (em grelha)?

Esse é o objetivo primordial desta dissertação: Tentar descobrir se utilizadores Spotify terão uma experiência melhorada ao usar a Aplicação Spotify proposta.

No entanto, para avaliar e validar o resultado final, será necessário fazer testes com utilizadores finais para comparar a sua experiência no Spotify com e sem a aplicação desenvolvida.

Desta forma, o desenvolvimento da aplicação será feito de forma iterativa, implementando as funcionalidades que o RAMA oferece, e quando esta estiver ao nível do RAMA, serão realizados testes cuidados com os utilizadores à medida que se melhora a implementação com o seu *feedback*.

## 1.3 Projeto

A aplicação a desenvolver será uma alternativa ao modo de descoberta do Spotify. No momento de escrita deste relatório, o modo de descoberta/recomendação de música do Spotify, comparativamente ao do Last.fm, é simples: apresenta recomendações em forma de grelha.

[4] http://youtube.com
[5] Application Programming Interface
[6] http://spotify.com
[7] https://developer.spotify.com/technologies/web-api

Desta forma, propõe-se uma representação visual em forma de grafo similar à do RAMA. A aplicação corre dentro do ambiente do Spotify como uma Aplicação Spotify. As suas principais funcionalidades serão: pesquisa de conteúdo, interação com o grafo e reproduzir música dos resultados da pesquisa. Estes são os requisitos mínimos que irão ser implementados.

Durante todo o desenvolvimento da aplicação, algumas das ferramentas a ser usadas serão:

**Spotify Desktop Client**
: O desenvolvimento de aplicações Spotify é feito de forma integrada no programa.

**Webkit Development Tools - webkit.org**
: A aplicação do Spotify foi desenvolvida com Webkit, e por isso, as aplicações Spotify também o são.

**Npmjs - npmjs.org**
: Gestor de pacotes de software e dependências.

**Gruntjs - gruntjs.com**
: Programa de gestão de tarefas automatizadas. Muito útil para testes, compilação e otimização de código.

**Arborjs - arborjs.org**
: Framework de javascript para desenho de grafos. Foi já utilizada no desenvolvimento do RAMA (existe sempre a possibilidade de se usar outra ferramenta substituta caso esta não for adequada).

## 1.4 Estrutura da Dissertação

Para além da introdução, este relatório contém mais 4 capítulos.

No capítulo 2, é descrito o estado da arte onde são apresentados trabalhos relacionados.

No capítulo 3, é explicado em detalhe em que consiste o projeto, com uma introdução ao ambiente de desenvolvimento de Aplicações Spotify, às tecnologias a serem usadas, à arquitetura e experimentação feita até ao momento.

No capítulo 4, é descrito com detalhe todo o plano de trabalho que esta dissertação vai seguir. Sempre que possível, será explicado com mais detalhe as tarefas mais importantes que estão previstas durante o desenvolvimento deste projeto.

No capítulo 5, são apresentadas conclusões sobre o planeamento do projeto até agora realizado.

# Capítulo 2

# Revisão Bibliográfica

## 2.1 Introdução

Neste capítulo será feita uma análise dos serviços já disponíveis que são relevantes para esta dissertação. Inicialmente, dar-se-á foco ao RAMA, para depois se comparar as suas funcionalidades com os projetos relacionados.

Esta dissertação foca-se mais na forma como se apresenta o conteúdo que se pretende recomendar ao utilizador, e não qual o conteúdo que é sugerido (não obstante da sua importância obviamente). No entanto, é quase impossível, no estudo do estado da arte, não se referir outros projetos que se focam também no conteúdo. Regra geral, os projetos que de seguida serão analisados, utilizam bases de dados externas, como o last.fm, para obter metadata que, convenientemente, também oferecem um tipo de recomendação de música com base numa pesquisa inicial. Tal não invalida que o tratamento dessa informação seja mal feito, e por isso, será feita uma pequena análise dos conteúdos sugeridos em cada um dos serviços.

## 2.2 RAMA - Relational Artist MAps

## 2.3 Projetos Relacionados

### 2.3.1 Liveplasma - liveplasma.com

O liveplasma.com é uma aplicação em flash que mostra relações de artistas de música em forma de grafo, para além de também permitir criar grafos com livros e filmes. Este não permite editar o grafo e, ao clicar num nó o grafo, é novamente gerado a partir desse nó.

Na figura 2.1 podemos ver o resultado de uma pesquisa. É possível ver a grelha com os álbums que o artista lançou, que redirecionam o utilizador para a Amazon[1] para comprar os álbums e um *mini-player* que começa a reproduzir uma música do artista diretamente do Youtube.

[1] http://amazon.com

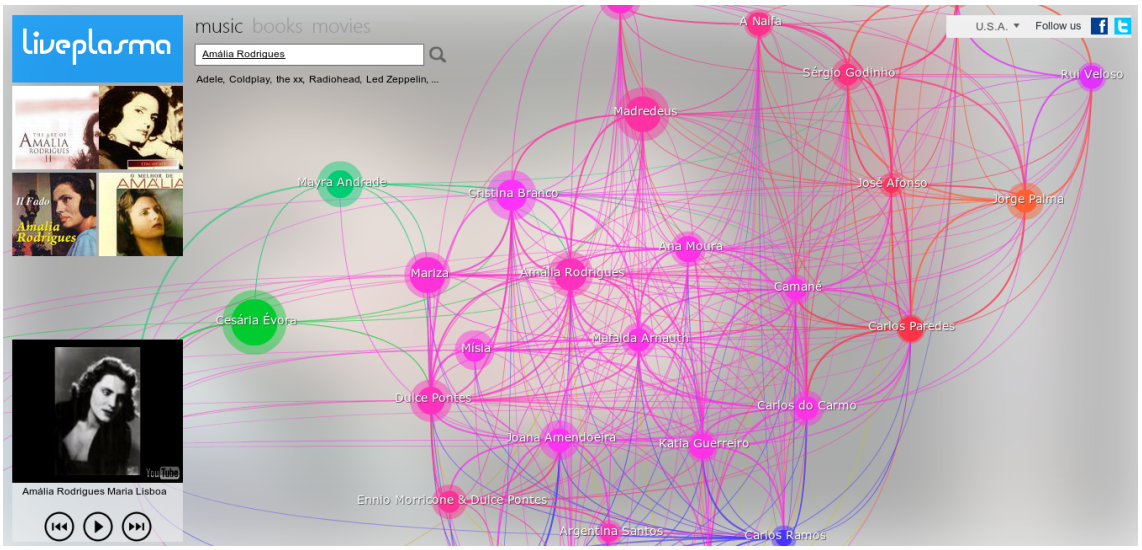



Figura 2.1: liveplasma: resultado da pesquisa "Amália Rodrigues". Canto superior esquerdo: álbums da artista; Canto inferior esquerdo: *mini-player* do youtube.

É possível controlar que músicas são reproduzidas de uma forma interessante: ao passar o rato por cima de um nó, aparece dois botões que permitem reproduzir música só do próprio artista (botão *only*) ou só de artistas parecidos (botão *similar*). É possível ver esses botões na figura 2.2

#### 2.3.1.1 Prós

Os aspetos interessantes desta ferramenta são:

- Links para compra dos álbums
- Reproduzir músicas de artistas semelhantes

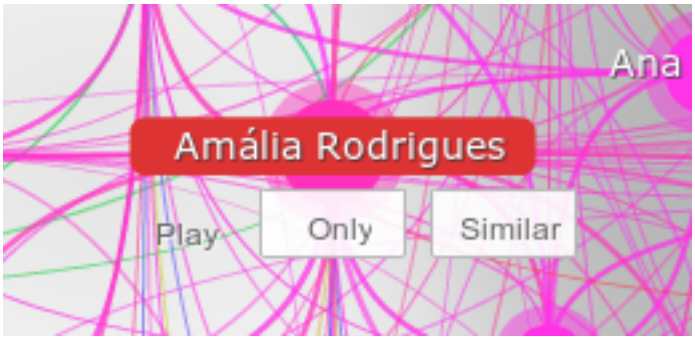



Figura 2.2: liveplasma: interface para reprodução de música. Botão *similar* reproduz músicas de artistas parecidos; Botão *only* só reproduz músicas do artista pesquisado.


Figura 2.3: Tuneglue: menu que aparece ao clicar num nó.

#### 2.3.1.2 Contras

O grafo desenhado é bastante confuso quando existem muitos nós com muitas ligações. Isto acontece quando existem muitos artistas semelhantes. Para além disso, são atribuídas cores aos nós que devem identificar o grau de parecença entre os artistas. No entanto não existe nenhum tipo de informação que explique qual o seu verdadeiro significado ao utilizador, assim como também não existe uma explicação das ligações entre os nós.

É também de notar que o tamanho dos nós é diretamente proporcional à popularidade dos artistas respectivos, mas mais uma vez, este tipo de informação não é dada ao utilizador.

Outra falha a apontar é o facto de não se conseguir distinguir o nó de pesquisa dos restantes resultados em 2.1 por exemplo.

#### 2.3.1.3 Resumo

Em suma, o liveplasma é usável, mas peca por ter muitas cores e ligações que tornam a experiência do utilizador ainda mais difícil do que a tradicional apresentação em lista ou grelha.

### 2.3.2 Tuneglue - audiomap.tuneglue.net

O Tuneglue é outro serviço do mesmo género (também desenvolvido em flash) que usa a base de dados do last.fm para recolher a informação dos artistas de música, assim como artistas relacionados.

Depois da pesquisa de um artista, por exemplo "Mariza", obtemos um grafo com apenas o nó de pesquisa. Ao clicar no nó é apresentado um menu com várias opções como se pode ver na figura 2.3.

A partir deste menu é possível ver uma das principais diferenças que o Tuneglue tem em relação ao liveplasma (2.3.1): a edição do grafo. É possível expandir, fixar e eliminar individualmente cada nó do grafo. Ao expandir o nó inicial de pesquisa, os nós novos estão apenas e só diretamente relacionados com o nó pai, como se pode ver na figura 2.4.

Ao contrário do Liveplasma, o Tuneglue não faz uma pesquisa recursiva.

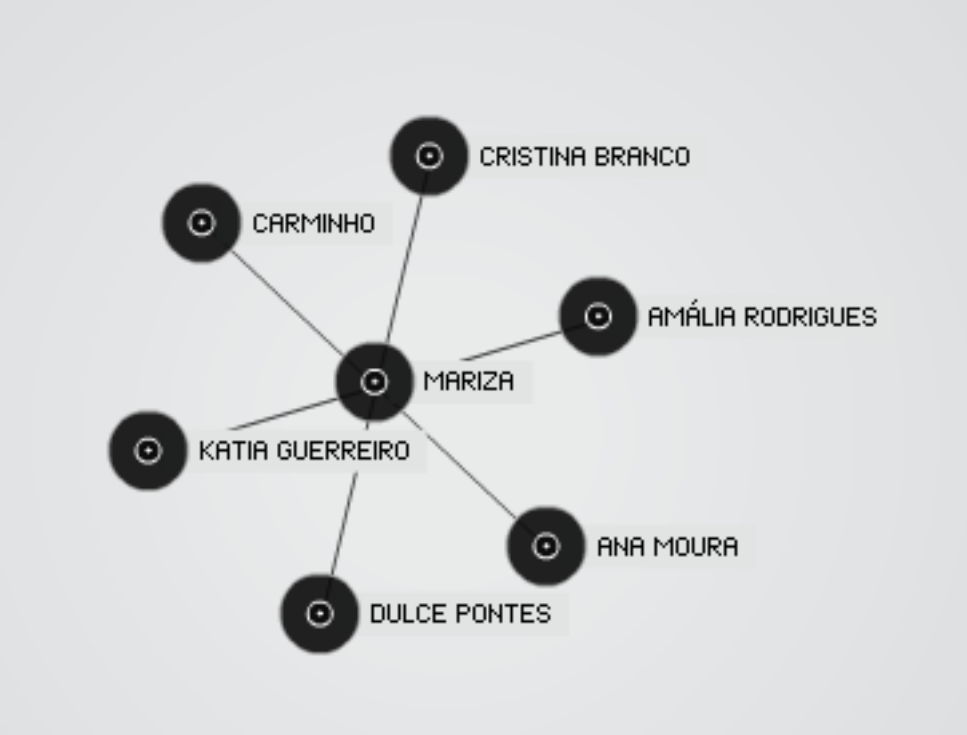



Figura 2.4: Tuneglue: grafo depois do primeiro nó ser expandido.

#### 2.3.2.1 Prós

Dá bastante liberdade ao utilizador, pois dá-lhe toda a responsabilidade na criação do grafo. O utilizador sente que todo o grafo foi criação sua e dissimula o utilizador a pensar que foi este que descobriu novos artistas ao invés de receber recomendações.

#### 2.3.2.2 Contras

Mais uma vez, o facto da ferramenta ser feita em flash não ajuda a que a interface seja intuitiva. Para além de pouco responsiva (o utilizador ao início pode-se sentir perdido por não saber o que fazer), é bastante uniforme, ou seja, não salienta diferenças entre cada artista, nem distingue as ligações entre eles.

#### 2.3.2.3 Resumo

Em suma, o Tuneglue é inteligente por dar poder ao utilizador, mas ao mesmo tempo não existe um limite nesse poder. E assim, é possível expandir o grafo até se tornar ilegível.

### 2.3.3 MusicRoamer - musicroamer.com

O MusicRoamer é outra ferramenta que permite explorar música nova. Tal como o Tuneglue, este permitir construir o grafo à medida que se expande cada nó.

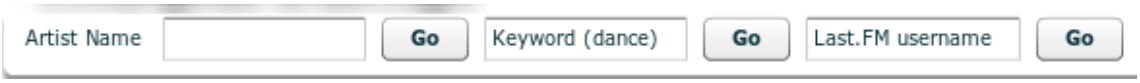



Figura 2.5: MusicRoamer: Várias opções de pesquisa. Por artista; por *keyword* e pelo perfil de utilizador do Last.fm

#### 2.3.3.1 Prós

Umas das funcionalidades interessantes do MusicRoamer são as várias opções de pesquisa (figura 2.5):

**Pesquisa por Artista**
Tipo de pesquisa mais utilizada

**Pesquisa por *Keyword***
Usar palavras-chave como géneros musicais para pesquisar livremente

**Pesquisa por perfil do Last.fm**
Esta pesquisa gera um grafo para cada artista (os mais ouvidos pelo utilizador)

Todas estas formas de pesquisa desenham um (ou mais) grafo(s) em que os nós são sempre artistas de música.

O que esta ferramenta trás de novo é a forma como apresenta os grafos. A figura 2.7 apresenta o resultado da pesquisa por Artista "Mariza". Imagens dos artistas são usadas para representar os nós, o que ajuda o utilizador a diferenciar os resultados.

A ferramenta também disponibiliza alguns parâmetros de personalização do grafo (figura 2.6) como Zoom, Tamanho da repulsão, imagem entre os nós e o número de artistas de música que deve expandir de um nó.

#### 2.3.3.2 Contras

Um problema do MusicRoamer é o facto de ser feito em flash, pois torna a interface menos natural e fluída. Para além disso, à medida que a profundidade do grafo vai aumentando, o grafo começa a ficar confuso e ilegível (figura 2.8). A linhas começam a se sobrepor e alguns nós ficam pouco legíveis.


Figura 2.6: MusicRoamer: Parâmetros de personalização do grafo

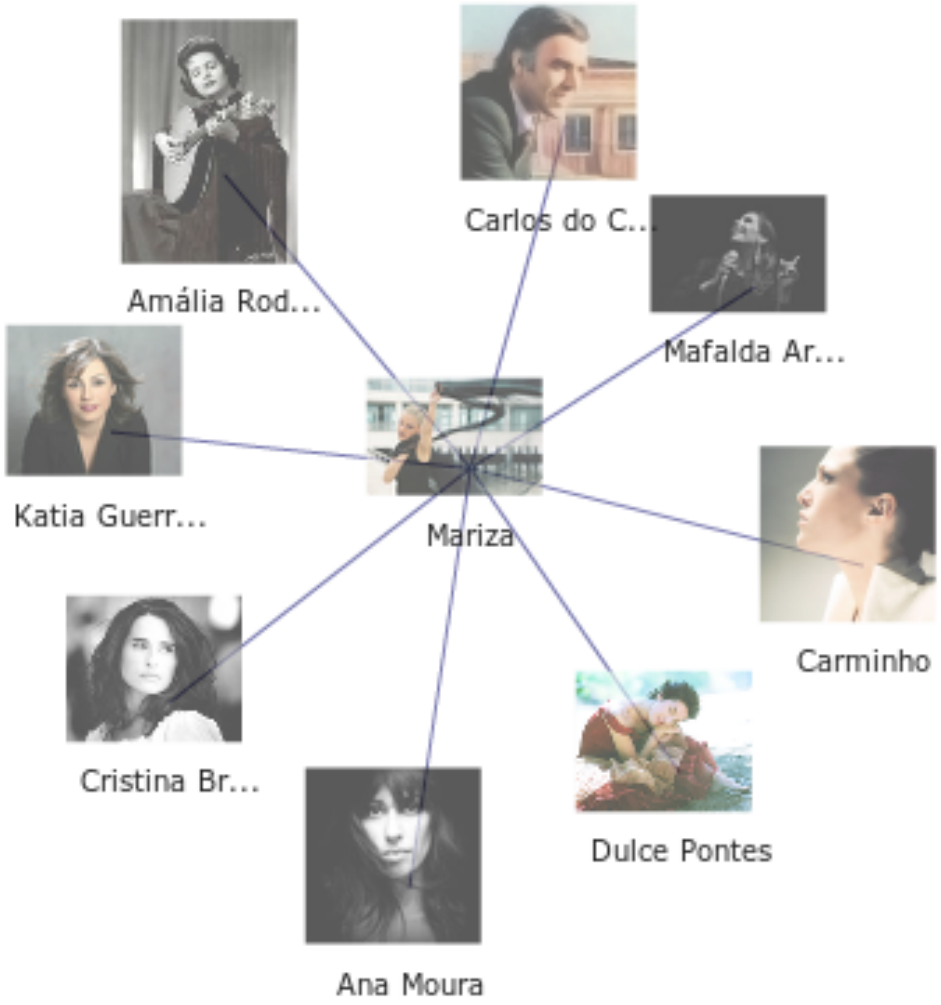



Figura 2.7: MusicRoamer: Representação visual do grafo de artistas

#### 2.3.3.3 Resumo

Apesar de um utilizador do MusicRoamer ter muita liberdade na criação do grafo, a sua apresentação global é fraca e pouco trabalhada esteticamente.

## 2.4 Resumo e Conclusões

Existem muitas outras ferramentas de descoberta de música. Apesar serem poucas as que usam esta representação visual em grafo, todas elas são importantes de se referir:

- liveplasma.com

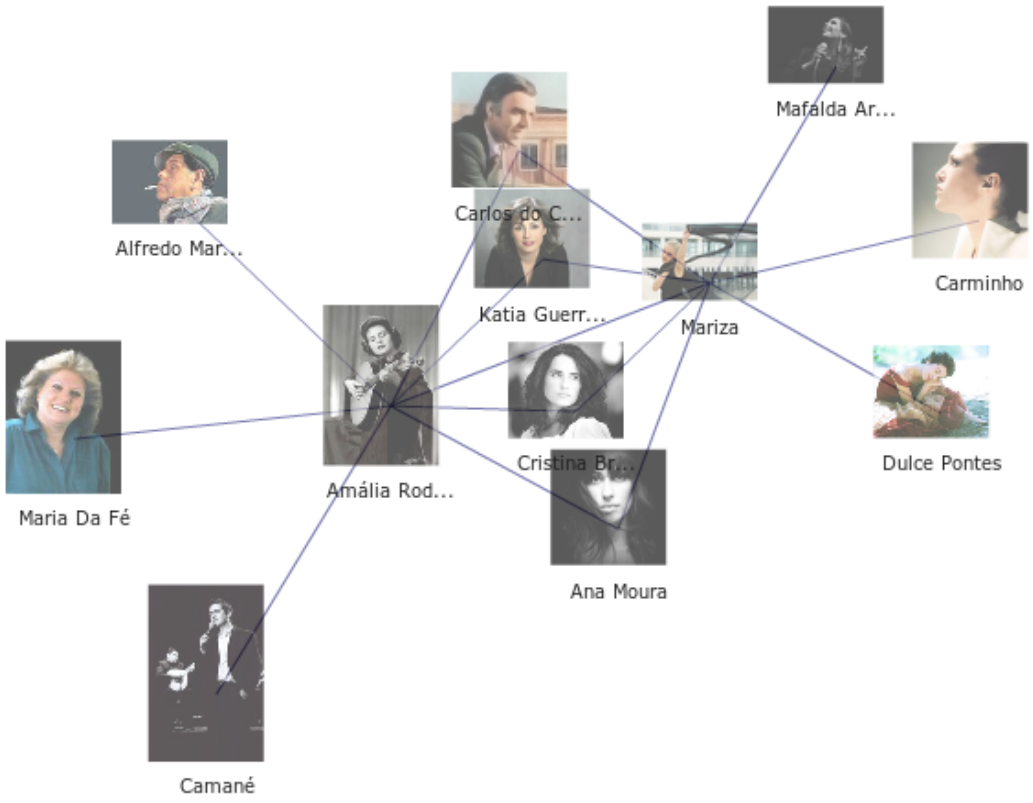



Figura 2.8: MusicRoamer: Grafo depois de expandir um nó

- audiomapa.tuneglue.net
- musicroamer.com
- discovr.info
- ifyoudig.net
- pitchfork.com
- hypem.com
- awdio.com
- 8tracks.com
- tastekid.com
- songza.com
- thesixtyone.com

- mog.com
- stereogum.com
- gigfi.com
- jango.com
- soundcloud.com
- grooveshark.com

Umas das primeiras lições que se tira dos exemplos dados é que quanto maior for o fator de ramificação de um grafo, mais confuso e saturado se torna. Não é um exagero dizer que para além de confuso, o grafo perde o seu propósito inicial de ajudar o utilizador na sua descoberta de música nova.

Uma forma de evitar este problema, será limitar o fator de ramificação a um máximo que não cause este problema.

# Capítulo 3

# Projeto

O principal objetivo desta dissertação, como foi referido no capítulo 1, é desenvolver um (ou mais) módulo(s) de software que contribuam para uma melhoria na descoberta e recomendação de música num ambiente integrado entre o RAMA e o Spotify, por forma a tirar partido da representação gráfica do grafo de artistas de música do RAMA e da qualidade do serviço de *Streaming* de música do Spotify.

Para tal, a proposta inicial desta dissertação consiste em desenvolver, no mínimo, um módulo que implemente uma das seguintes funcionalidades:

1. Integrar o serviço de *streaming* de música do Spotify **no RAMA**
2. Integrar informação de um utilizador Spotify **no RAMA**
3. Melhorar design e funcionalidades **do RAMA**
4. Integrar a visualização de grafos de artistas de música **numa Aplicação Spotify**
5. Integrar o módulo de criação de *playlists* do RAMA **numa Aplicação Spotify**
6. Integrar alguns dos módulos acima referidos **numa aplicação móvel**

As três primeiras funcionalidades (1, 2 e 3) focam-se em melhorar o serviço do RAMA, usando API's do Spotify, ou seja, integrar o Spotify dentro do RAMA. Por outro lado, as funcionalidades 4 e 5 têm como objetivo integrar o RAMA dentro do Spotify, através de uma Aplicação Spotify, que funciona como plugin do programa principal do Spotify. A última funcionalidade (6) teria de implementar algumas das anteriores num Sistema Operativo Móvel (Android, iOS ou Windows Phone).

Este capítulo procura analisar todas as condicionantes que afetam a escolha dos módulos a desenvolver, e em que ambientes estes se encaixam melhor (Aplicação Spotify, aplicação móvel ou RAMA).

Inicialmente será explorado o ambiente de desenvolvimento que o Spotify disponibiliza, ou seja, que tecnologias tem disponíveis para *developers*. De seguida serão analisadas quais dessas tecnologias assentam melhor em cada um dos módulos propostos a desenvolver, através de experimentações feitas, e quando necessário, será descrito um possível esquema de arquitetura por forma a facilitar a explicação do problema.

No final deste capítulo, deve ficar claro quais serão os módulos de software a desenvolver, que tecnologias irão ser usadas e qual o esquema geral da sua arquitetura. O produto final deve de ir ao encontro do objetivo de contribuir para uma melhoria na descoberta e recomendação de música num ambiente relacionado com o RAMA.

## 3.1 Spotify

O Spotify é um serviço de *streaming* de música que permite ouvir, através de uma ligação de Internet, qualquer música que o Spotify possua no seu catálogo.

### 3.1.1 Ferramentas de Desenvolvimento

No momento de escrita deste relatório, o Spotify tem disponível um conjunto de ferramentas[1] para desenvolver módulos de software que podem estar embebidos nas mais diversas aplicações (*third-party applications*) ou então dentro do *Spotify Desktop Client*.

Existem quatro ferramentas de desenvolvimento, cada uma delas com o seu propósito e utilidade.

#### 3.1.1.1 Spotify Apps

Serve para desenvolver Aplicações Spotify[2] que são usadas pelos utilizadores Spotify dentro do *Spotify Desktop Client*. Estas são aplicação *HTML5*[3].

Na figura 3.1 é possível ver o aspeto do *Spotify Desktop Client*. Na barra lateral esquerda, dentro do separador *Apps*, aparece a lista de aplicações já instaladas, assim como o *App Finder*, que permite procurar e instalar aplicações com apenas um clique.

Na figura 3.2 está aberta a aplicação da Last.fm. É possível ver que as Aplicações Spotify têm apenas um espaço reservado embutido no *Spotify Desktop Client*.

Para o seu desenvolvimento destas aplicações são disponibilizadas duas *frameworks*: *API Framework*[4] e *Views Framework*[5]. A primeira fornece uma interface para recolher metadados de artistas, álbuns e músicas e controlar o reprodutor de música. A segunda fornece componentes de design como botões, listas, abas, entre outros, para o desenvolvimento da aplicação.

Para desenvolver os módulos 4 e 5 esta é a ferramenta mais apropriada.

[1] http://developer.spotify.com/technologies
[2] https://developer.spotify.com/technologies/apps
[3] http://www.w3.org/TR/html5/
[4] https://developer.spotify.com/docs/apps/api/1.0/
[5] https://developer.spotify.com/docs/apps/views/1.0/

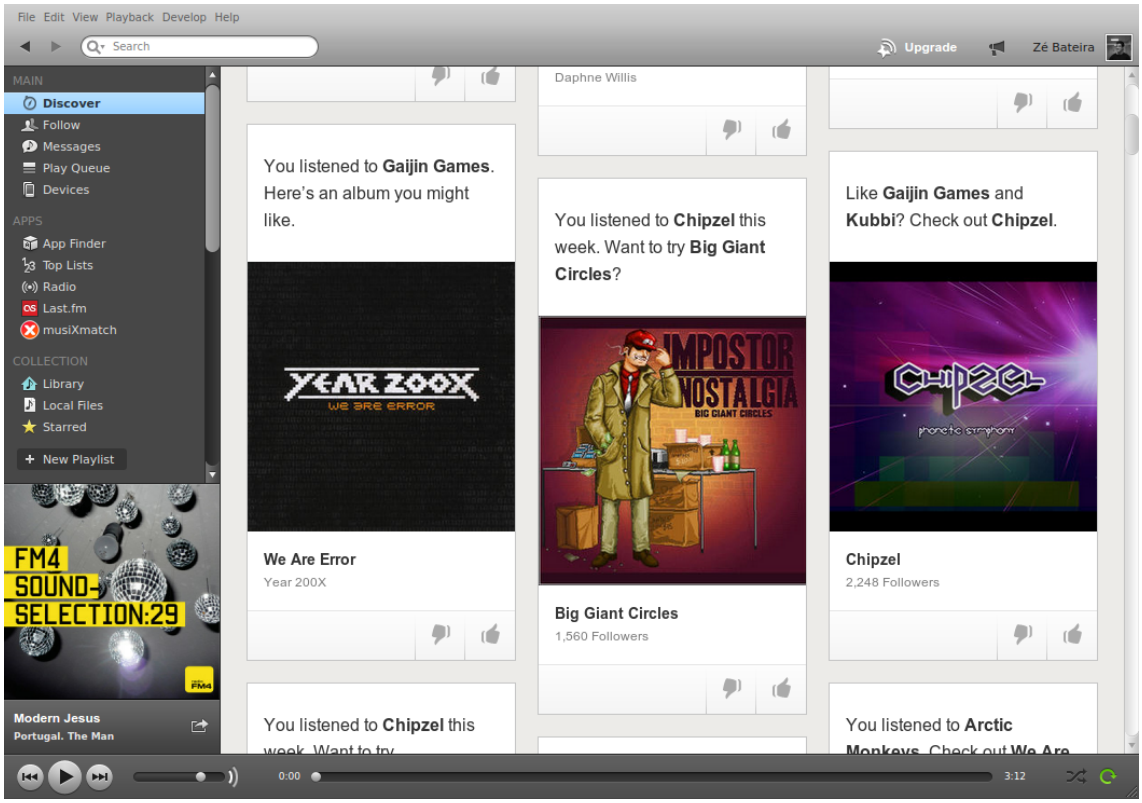



Figura 3.1: Spotify: interface do modo de descoberta do *desktop client*

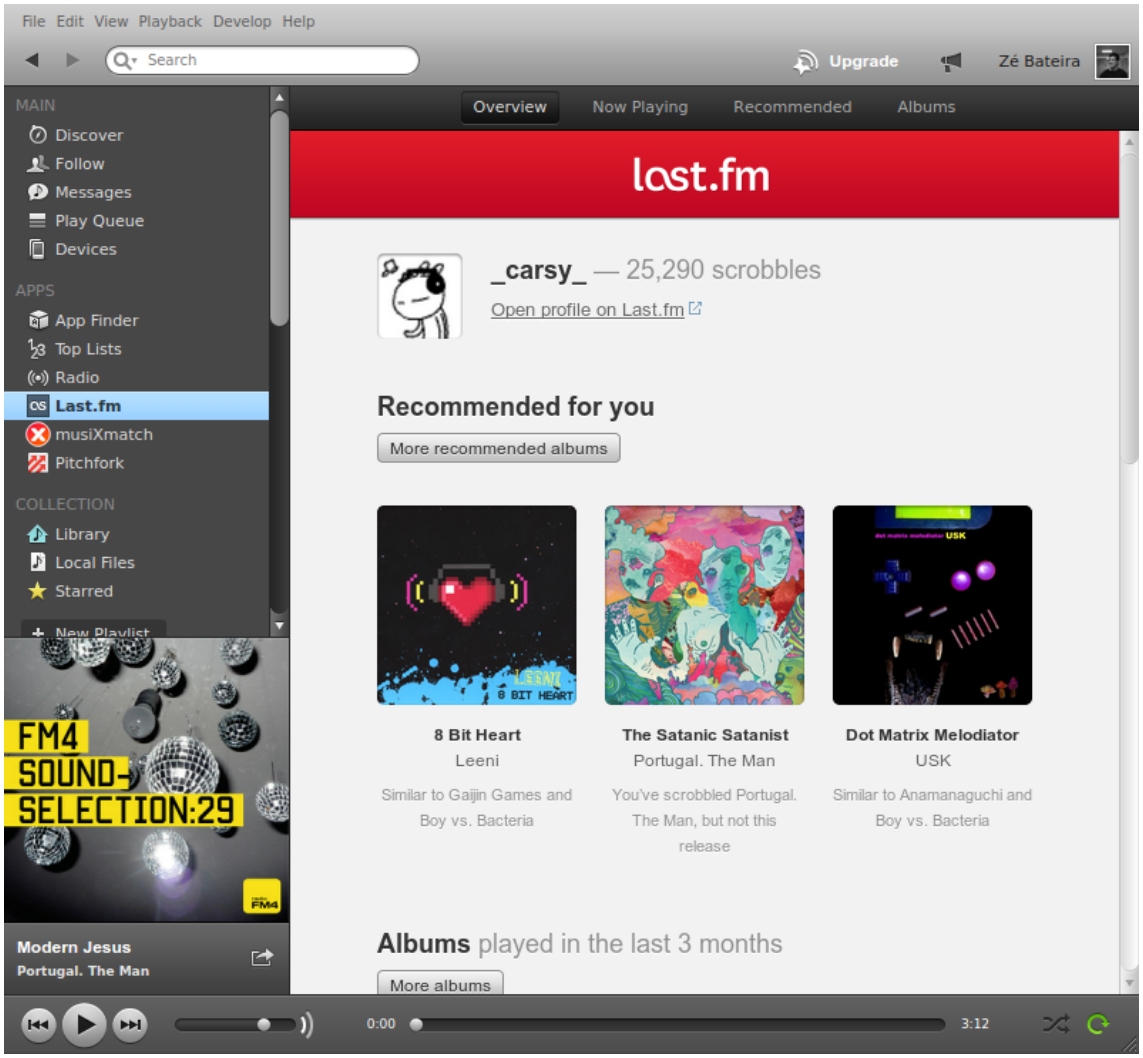



Figura 3.2: Spotify: Aplicação Last.fm aberta no *Spotify Player*

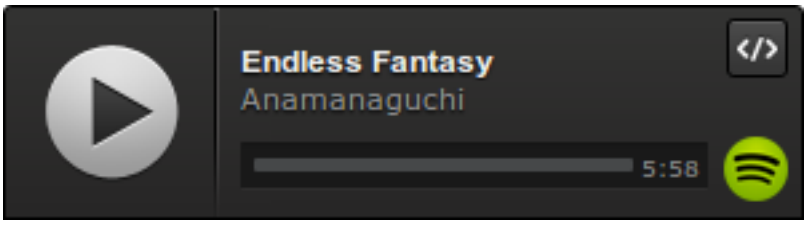



Figura 3.3: Spotify: *Play Button* pode ser embebido em *websites*.

#### 3.1.1.2 Spotify Widgets

As Widgets[6] são pequenos componentes que se podem embeber em *websites*. No momento da escrita deste relatório existem dois componentes: *Play Button* (3.3) e *Follow Button* (3.4).

No entanto, existe algumas limitações no uso destas componentes. No Spotify, apenas utilizadores que tenham criado conta no serviço Spotify é que podem usar o mesmo. O mesmo também se aplica a estas *widgets* - apesar de estas existirem numa aplicação externa ao Spotify, apenas utilizadores Spotify podem usá-las. Esta limitação pode fazer sentido para o *Follow Button*, mas o *Play Button* torna-se inútil para utilizadores que não usem o Spotify. Outro problema surge quando a música do *Play Button* não está disponível no País em que o utilizador está.

Estas *widgets* apenas servem de hiperligação ou ao *Player* do Spotify ou ao *WebPlayer* do Spotify. Na realidade, usando estas *widgets*, o stream de música do Spotify é sempre reproduzido dentro do ambiente do Spotify, e nunca em aplicações externas.

Para embeber as *widgets* apenas é necessário introduzir um elemento *iframe* no código *HTML* fonte do *website*:

```
1 <iframe src="https://embed.spotify.com/?uri=spotify:track:1
      EsdqTsiQPaUJ82iy7KfS1"
2         frameborder="0"
3         width="300"
4         height="380">
5 </iframe>
```

Listing 3.1: Código *HTML* para embeber um *Play Button*

[6] https://developer.spotify.com/technologies/widgets

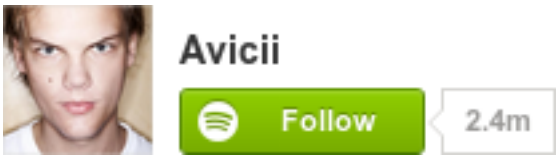



Figura 3.4: Spotify: *Follow Button* permite seguir um artista.

As *widgets* seriam úteis para desenvolver os módulos 1 e 3.

#### 3.1.1.3 Libspotify SDK

Libspotify SDK[7] é uma API que permite adicionar os serviços do Spotify em aplicações externas. No entanto, existem algumas limitações para os utilizadores destas aplicações.

Existem, dois tipos de conta a que o utilizador pode subscrever: conta grátis e conta *premium*. Como foi referido anteriormente (3.1.1.2) apenas utilizadores Spotify podem interagir com qualquer componente do Spotify, dentro ou fora das aplicações nativas do mesmo. Libspotify fornece uma interface que permite a um utilizador fazer *login* no Spotify em aplicações externas por forma a poder ouvir música do Spotify, criar playlists e outras funcionalidades. No entanto, os únicos utilizadores que pode fazer *login* nestas aplicações que usam Libspotify, são utilizadores *premium*. Para além de que, os *developers* da própria aplicação também precisam de ser utilizadores *premium*.

Neste sentido, uma aplicação que, para funcionar, necessita de que o utilizador, para além de possuir uma conta Spotify, também pague uma subscrição mensal *premium*, é uma aplicação bastante restritiva.

Esta ferramenta pode ser usada para desenvolver os módulos 1, 2 e 6.

#### 3.1.1.4 Metadata API

A *Metadata API*[8] disponibiliza publicamente informação de músicas, álbuns e artistas da Base de dados do Spotify.

Através de pedidos HTTP é possível obter informação da base de dados do Spotify. Existe dois tipos de pedidos que esta API disponibiliza: *search*[9] e *lookup*[10]. Para obter informação detalhada de, por exemplo, um artista, é necessário saber o deu identificador único. Esse identificador é um *URI* da forma:

`spotify:artist:<artist_id>`, onde *artist_id* é um identificar único.

Exemplo:

`spotify:artist:65nZq8l5VZRG4X445F5kmN`, é o identificador único da fadista "Mariza".

Também existem identificadores únicos para álbuns:

`spotify:album:5d1LpIPmTTrvPltx26TlEU` (álbum "Fado Tradicional"de "Mariza")

e para faixas de música:

`spotify:track:2vqYasauhDLVjTt7CGWK6y` (música "Fado Vianinha"do mesmo álbum)

[7] https://developer.spotify.com/technologies/libspotify
[8] https://developer.spotify.com/technologies/web-api
[9] https://developer.spotify.com/technologies/web-api/search
[10] https://developer.spotify.com/technologies/web-api/lookup

Para obter este *URI* é preciso interrogar a base de dados com um método de pesquisa. Para isso, usa-se o *search*.

***Search***

O *URL* base de utilização é:

http://ws.spotify.com/search/1/album, para pesquisa de álbuns.

Se se pretender pesquisar Artistas, usa-se *artist*, se se pretender pesquisar Faixas de música, usa-se *track*.

Exemplos:

http://ws.spotify.com/search/1/album?q=foo
http://ws.spotify.com/search/1/artist.json?q=red+hot

O resultado da *query*, por defeito, tem o format *XML*. No entanto, também se pode especificar o formato *JSON* (como no segundo exemplo).

Dada a *query*:
http://ws.spotify.com/search/1/artist.json?q=camane (fadista "Camané")

Obtém-se o resultado:

```
{
    "info": {
        "num_results": 2,
        "limit": 100,
        "offset": 0,
        "query": "camane",
        "type": "artist",
        "page": 1
    },
    "artists": [
        {
            "href": "spotify:artist:3MLPFTe4BrpEV2eOVG0gLK",
            "name": "Camane",
            "popularity": "0.27"
        },
        {
            "href": "spotify:artist:5Gwulm1LfURW7dbZD1V3zX",
            "name": "Sergio Godinho/Camane/Carlos Do Carmo",
            "popularity": "0"
        }
    ]
}
```

Listing 3.2: Os resultados são ordenados pelo atributo "popularity"

***Lookup***

Depois de obtido o *URI* identificador, é possível obter mais informações de um conteúdo usando o *lookup*.

A seguinte *query*:
http://ws.spotify.com/lookup/1/.json?uri=spotify:artist:3MLPFTe4BrpEV2eOVG0gLK

Retorna:

```
{
    "info": {
        "type": "artist"
    },
    "artist": {
        "href": "spotify:artist:3MLPFTe4BrpEV2eOVG0gLK",
        "name": "Camane"
    }
}
```

Listing 3.3: Resultado do *lookup* do fadista "Camané"

Esta API seria bastante útil para desenvolver qualquer um dos seis módulos propostos. Aliás, até complementa as *Widgets* e o *Libspotify SDK*.

### 3.1.2 Experimentações Feitas

Numa primeira experiência com as ferramentas, foi criado um pequeno *website* que permite pesquisar e ouvir Música do Spotify usando a *Metadata API* e *Spotify Widgets*:

http://carsy.github.io/spotify-playground

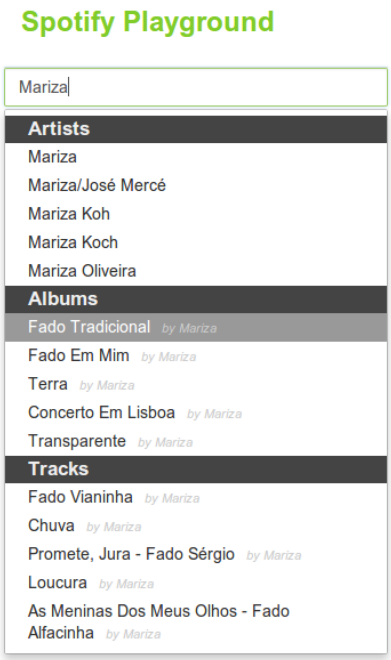
Na figura 3.5 é possível ver o resultado de uma pesquisa, e a *Widget Play Button* com o resultado selecionado da pesquisa.

Verificou-se que as duas ferramentas estão bem documentadas e em constante atualização.

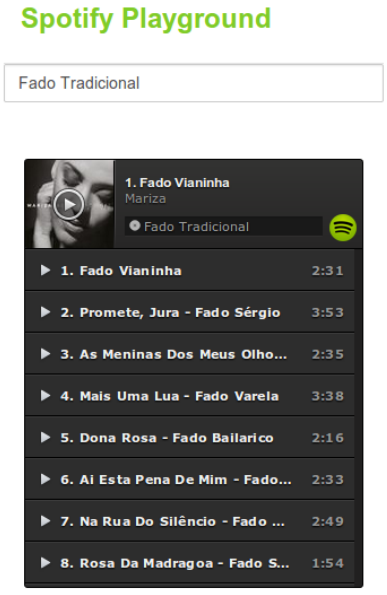
Outra experiência foi realizada para verificar se é possível usar o elemento *canvas* numa Aplicação Spotify. Isto é necessário pois será a única forma de poder desenhar graficamente o grafo. Para isso foi apenas necessário criar uma aplicação com o seguinte código fonte:

```
<iframe src="http://rama.inescporto.pt/app" frameborder="0"></iframe>
```

Listing 3.4: Elemento *iframe* que embebe o *website* do RAMA na aplicação




(a) Resultado da pesquisa "Mariza"




(b) Depois de selecionado o álbum "Fado Tradicional"aparece o *Play button* com as faixas do álbum.

Figura 3.5: Experiência com *Metadata API* e *Play Button Widget* (código fonte: `github.com/carsy/spotify-playground`)

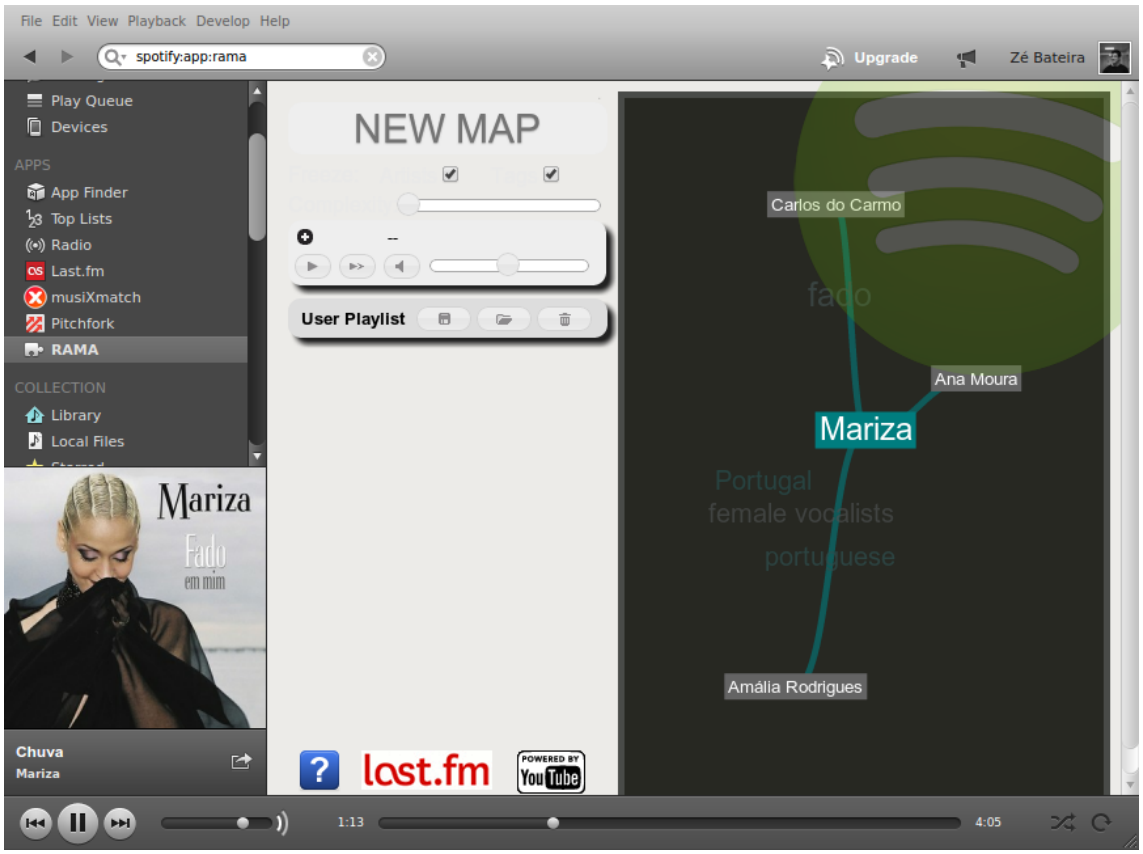



Figura 3.6: *Website* do RAMA embebido numa Aplicação Spotify

Desta forma, é possível embeber o RAMA na Aplicação Spotify (que usa o elemento *canvas* para desenhar o grafo). Resultado final na figura 3.6.

Apesar de *iframes* serem suportadas, existem outros componentes que não o são. A aplicação não é usável, pois não permite, por exemplo, reproduzir automaticamente faixas de artistas.

No entanto existe uma forma de testar quais os elementos de *HTML5* suportados, usando uma aplicação interna do Spotify. Na figura 3.7 é possível ver que o elemento *canvas* é suportado a cem por cento.

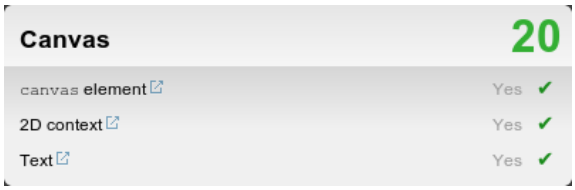



Figura 3.7: Resultado do teste do elemento *canvas*

### 3.1.3 Conclusão

A prova de conceito desenvolvida (3.6) demonstrou-se a mais indicada para o objetivo final de criar um ambiente integrado entre o Spotify e o RAMA.

Assim, os módulos a serem desenvolvidos são 4 e 5.

## 3.2 Tecnologias

As seguintes tecnologias serão utilizadas nas fase de desenvolvimento, testes e optimização da Aplicação Spotify.

### 3.2.1 *Spotify Desktop Client*

O desenvolvimento de Aplicações Spotify é feito de forma integrada no programa.

Para abrir uma Aplicação Spotify, localmente, escreve-se o seguinte na barra de pesquisa: spotify:app:rama

Onde *rama* deve ser o identificador da aplicação declarado no ficheiro *manifest.json*[11].
Exemplo de ficheiro *manifest.json*:

```
{
  "AppName": {
    "en": "RAMA"
  },
  "BundleIdentifier": "rama",
  "AppDescription": {
    "en": "RAMA: Relational Artist MAps"
  },
  "AcceptedLinkTypes": [
    "playlist"
  ],
  "BundleType": "Application",
  "BundleVersion": "0.2",
  "DefaultTabs": [
    {
      "arguments": "index",
      "title": {
        "en": "Home"
      }
    }
  ],
  "Dependencies": {
    "api": "1.10.2",
    "views": "1.18.1"
  },
```

[11] ficheiro situado na *root* da pasta do projeto

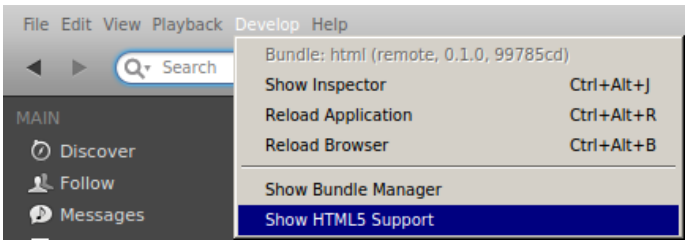



Figura 3.8: Menu *Develop*

```
26   "SupportedDeviceClasses": ["Desktop"],
27   "SupportedLanguages": [
28     "en"
29   ],
30   "VendorIdentifier": "pt.inescporto"
31 }
```

Listing 3.5: manifest.json: *BundleIdentifier* é o identificador da aplicação; *Dependencies* declara as dependências das API's necessárias ao desenvolvimento.

Existem outras opções úteis a que se pode aceder usando a tab *Develop* (3.8). A opção "Show Inspector"abre a janela *Webkit Development Tools* (3.2.2)

### 3.2.2 Webkit Development Tools - webkit.org

A partir do *webkit*, tem-se acesso a várias ferramentas úteis para o desenvolvimento *web* (3.9). A mais importantes são:

**Inspector** Permite inspecionar e editar o código *HTML* e *CSS* da aplicação diretamente (3.9).

**Network** Permite, por exemplo, ver o tempo que cada componente da aplicação demorou a carregar (uma imagem ou um ficheiro *css*) (3.10).

**Profile** Permite identificar que partes do código *javascript* são as mais frequentemente executadas (3.11).

**Audit** Ajuda a perceber quantos recursos estão a ser descarregados desnecessariamente, como por exemplo, regras de *CSS* que não estão a ser usadas (3.12).

**Console** Muito útil para *debug* de *javascript*.

### 3.2.3 Npmjs - npmjs.org

Gestor de pacotes de software e dependências. Para usar *npm* é necessário um ficheiro de configuração *package.json* que permite identificar quais os pacotes de que a aplicação depende, e as suas versões.
Exemplo:

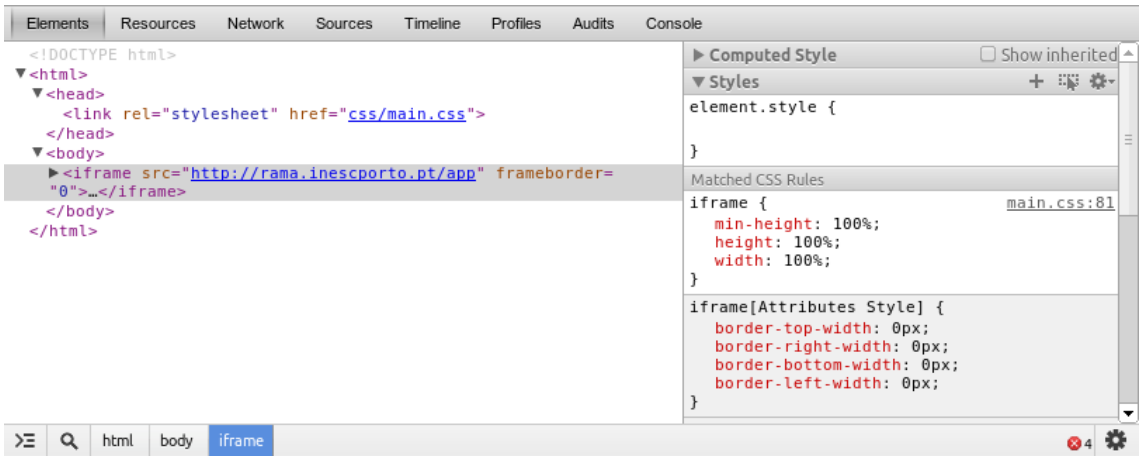



Figura 3.9: Webkit: Vista da tab *Inspector*. Outas ferramentas disponíveis (tabs): *Resources, Network, Sources, Timeline, Profiles, Audits* e *Console*.

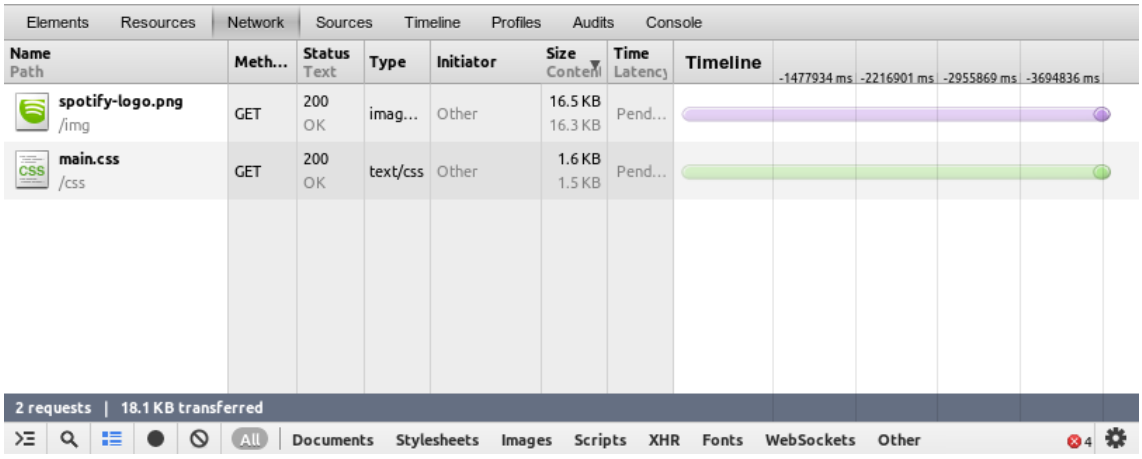



Figura 3.10: Webkit Network

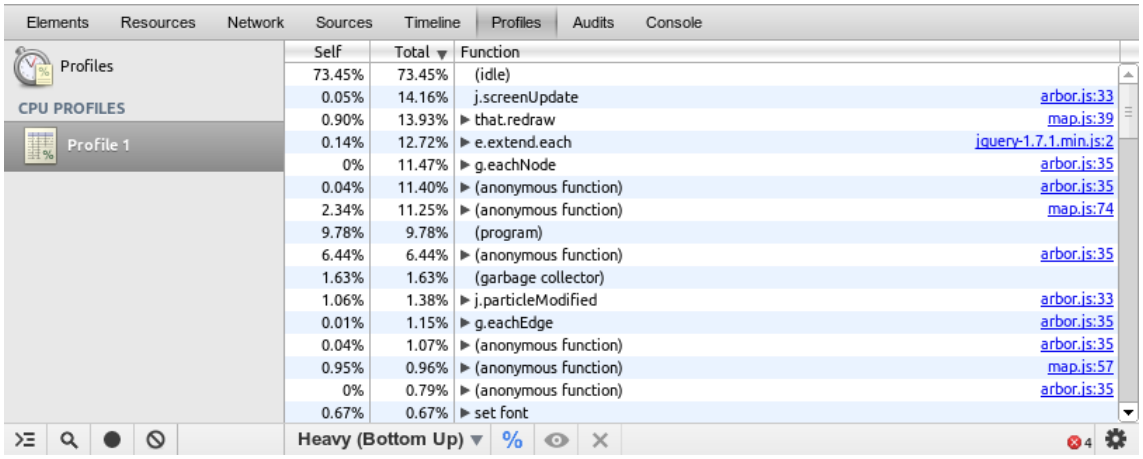



Figura 3.11: Webkit Profile: É possível ver que a renderização do grafo é o que ocupa mais tempo de processamento como esperado. No entanto, existe uma parte de *JQuery* que ocupa 12.72% do tempo de processamento, o que pode indicar um possível ponto de melhoria de performance.

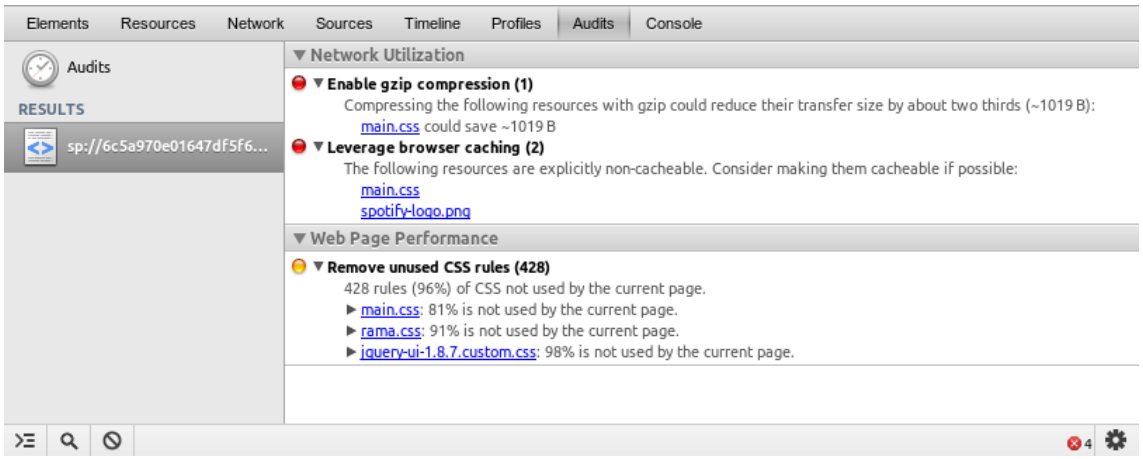



Figura 3.12: Webkit Audit: 96% do código *CSS* não está a ser usado, sendo por isso, um ponto de melhoria reduzir a quantidade de informação descarregada.

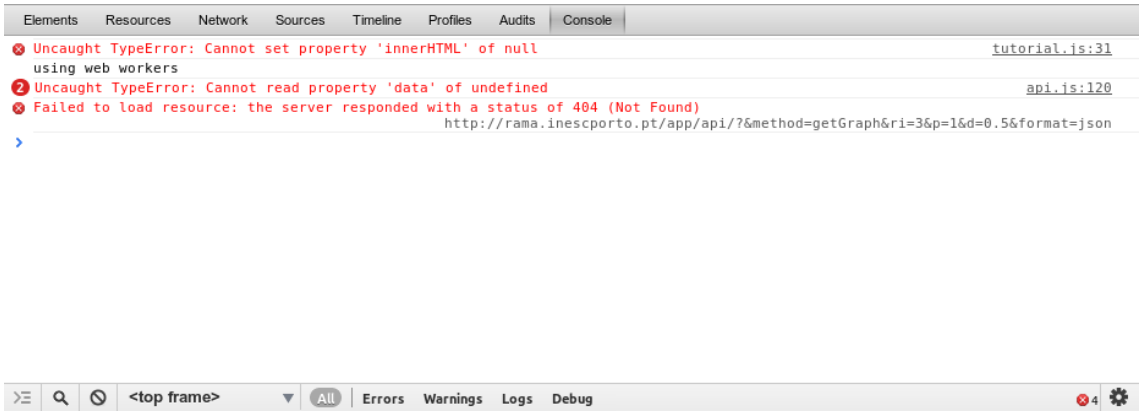



Figura 3.13: Webkit Console: Erros de *Javascript* aparecem destacados para chamar a atenção.

```
{
  "name": "RAMA",
  "devDependencies": {
    "grunt": "~0.4.2",
    "grunt-contrib-jshint": "*",
    "grunt-contrib-jasmine": "*",
    "grunt-contrib-watch": "*"
  },
  "version": "0.1.0"
}
```

Listing 3.6: *package.json*: ao indicar a versão com "*", significa que se deve usar sempre a mais recente.

### 3.2.4 Gruntjs - gruntjs.com

Programa de gestão de tarefas automatizadas. Muito útil para testes, compilação e otimização de código. É possível por exemplo, quando qualquer parte do código mudar, a aplicação automaticamente atualiza com as mudanças mais recentes, sem ser preciso refrescar manualmente a aplicação.

### 3.2.5 Arborjs - arborjs.org

Framework de javascript para desenho de grafos. Foi já utilizada no desenvolvimento do RAMA (existe sempre a possibilidade de se usar outra ferramenta substituta caso esta não for adequada).

## 3.3 Arquitetura

Esquema geral da arquitetura da Aplicação.

## 3.4 Resumo e Conclusões

A escolha final do módulo a desenvolver é a Aplicação Spotify.

# Capítulo 4

# Plano de Trabalho

Neste capítulo serão explicadas as diferentes fases de desenvolvimento desta dissertação, atribuídos pontos de esforço a todas as tarefas de cada fase e definida a calendarização das mesmas fases.

## 4.1 Fases do Projeto

Cada fase terá um conjunto de tarefas associadas, e cada uma deles tem um quantificador de *esforço* (escala *Fibonacci* de 1 a 8) por forma a ajudar a compreender algumas das distribuições de tempo de trabalho para cada tarefa. Para identificar uma tarefa, usou-se a nomenclatura:

<*fase*>.<*tarefa*>

Exemplo: Tarefa 3.2 é a segunda tarefa da terceira fase do projeto.

### 4.1.1 Fase 1 - Desenho da Aplicação

Numa primeira fase, serão quantificadas as funcionalidades a implementar por forma a desenhar bem o espaço da aplicação.

**Tarefa 1.1 - Estudo detalhado das funcionalidades**

Perceber quais as funcionalidades que vai ter mais destaque na aplicação.

*Esforço: 3*

**Tarefa 1.2 - Desenho global**

Especificação do *layout* de todas as vistas da aplicação.

*Esforço: 5*

### 4.1.2 Fase 2 - Mapeamento de Metadados Spotify em Last.fm

O objetivo desta segunda fase da dissertação é conseguir fazer corresponder o conjunto de metadados da base de dados do Spotify na da Last.fm.

Para isso, esta fase tem três tarefas associadas:

**Tarefa 2.1 - Recolha de Informação**

Recolher informação relevante que ajude a perceber que tipos de metadados similares existem entre as duas bases de dados.

*Esforço: 2*

**Tarefa 2.2 - Módulo de mapeamento**

Desenvolvimento de um módulo capaz de fazer esse mesmo mapeamento de uma forma modelar. Deve tentar compilar a maior quantidade de metadados de cada parte numa entrada.

*Esforço: 5*

**Tarefa 2.3 - Pesquisa de um Artista** Desenvolvimento da funcionalidade de pesquisa de um artista, usando o módulo criado anteriormente.

*Esforço: 3*

### 4.1.3 Fase 3 - Criação e Edição do grafo

Esta é a fase crítica da dissertação, pois contém as tarefas com maior classificação de esforço.

**Tarefa 3.1 - Representação em Grafo**

Representação abstrata das relações dos artistas de música em grafo. Esta tarefa depende bastante da tarefa 2.3 no sentido em que ainda é incerto, se é possível utilizador a metodologia utilizada pelo RAMA para gerar o grafo.

*Esforço: 8*

**Tarefa 3.2 - Desenho gráfico do grafo**

Desenho do grafo usando uma ferramenta de renderização de grafos 2D3.2.5. Se o desempenho da ferramenta impedir uma boa experiência de utilizador, será necessário investigar mais e procurar outras ferramentas que permitam uma melhor performance.

*Esforço: 8*

**Tarefa 3.3 - Edição do grafo**

Disponibilizar funcionalidades de edição do grafo desenhado como eliminar, expandir e mover nó.

*Esforço: 3*

**Tarefa 3.4 - Parâmetros do grafo**

Como funcionalidades mais avançadas, mostrar parâmetros editáveis do grafo. Não esquecer de limitar alguns parâmetros por forma a evitar comportamentos erráticos do grafo.
*Esforço: 3*

### 4.1.4 Fase 4 - Reprodução de Música

Nesta quarta fase da dissertação, serão implementadas mais funcionalidades essenciais à aplicação.

**Tarefa 4.1 - Reproduzir música de artista**

Permitir selecionar um nó e reproduzir as música mais populares desse artista.
*Esforço: 3*

**Tarefa 4.2 - Gerar *Playlists***

Gerar uma *playlists* a partir de um grafo.
*Esforço: 3*

**Tarefa 4.3 - Guardar *Playlists***

Permitir ao utilizador guarda a *playlists* gerada.
*Esforço: 1*

**Tarefa 4.4 - Seguir Artista**

Seguir artista de qualquer nó do grafo.
*Esforço: 2*

### 4.1.5 Fase 5 - Avaliação e Validação

Esta fase final do projeto irá contemplar uma avaliação da aplicação utilizando o *feedback* de utilizadores que irão experimentar a mesma, por forma a validar o objetivo desta dissertação: melhoria de uma experiência musical de um utilizador Spotify.

**Tarefa 5.1 - Recolha de dados de utilização da Aplicação**

Os utilizadores inicialmente irão mostrar os seus hábitos de pesquisa de nova música. De seguida, ser-lhes-á introduzida a aplicação desenvolvida para se ficar com uma perceção do uso que lhe é dada.
*Esforço: 3*

**Tarefa 5.2 - Análise dos Dados recolhidos**

Tirar conclusões dos dados recolhidos por forma a tirar conclusões sobre a forma de utilização da aplicação.
*Esforço: 5*

**Tarefa 5.3 - Melhorias na Aplicação**

Melhorias na aplicação de acordo com o *feedback* dos utilizadores.

*Esforço: 3*

## 4.2 Calendarização

Na figura 4.1 é possível ver a calendarização do plano de trabalho para esta dissertação. É de notar que a produção da documentação (dissertação, artigo e apresentação) não fazem parte das fases de desenvolvimento, no entanto, estão definidas na calendarização.

## 4.3 Resumo

O planeamento desta dissertação foi feita de forma a que o grau de esforço das fases de desenvolvimento cresça ao longo do tempo. Assim, o desenvolvimento inicial será mais suave e natural, camuflando o aumento do grau de complexidade das seguintes fases.

Tentou-se distribuir mais tempo para fase mais críticas, e menos tempo para fases menos prioritárias.

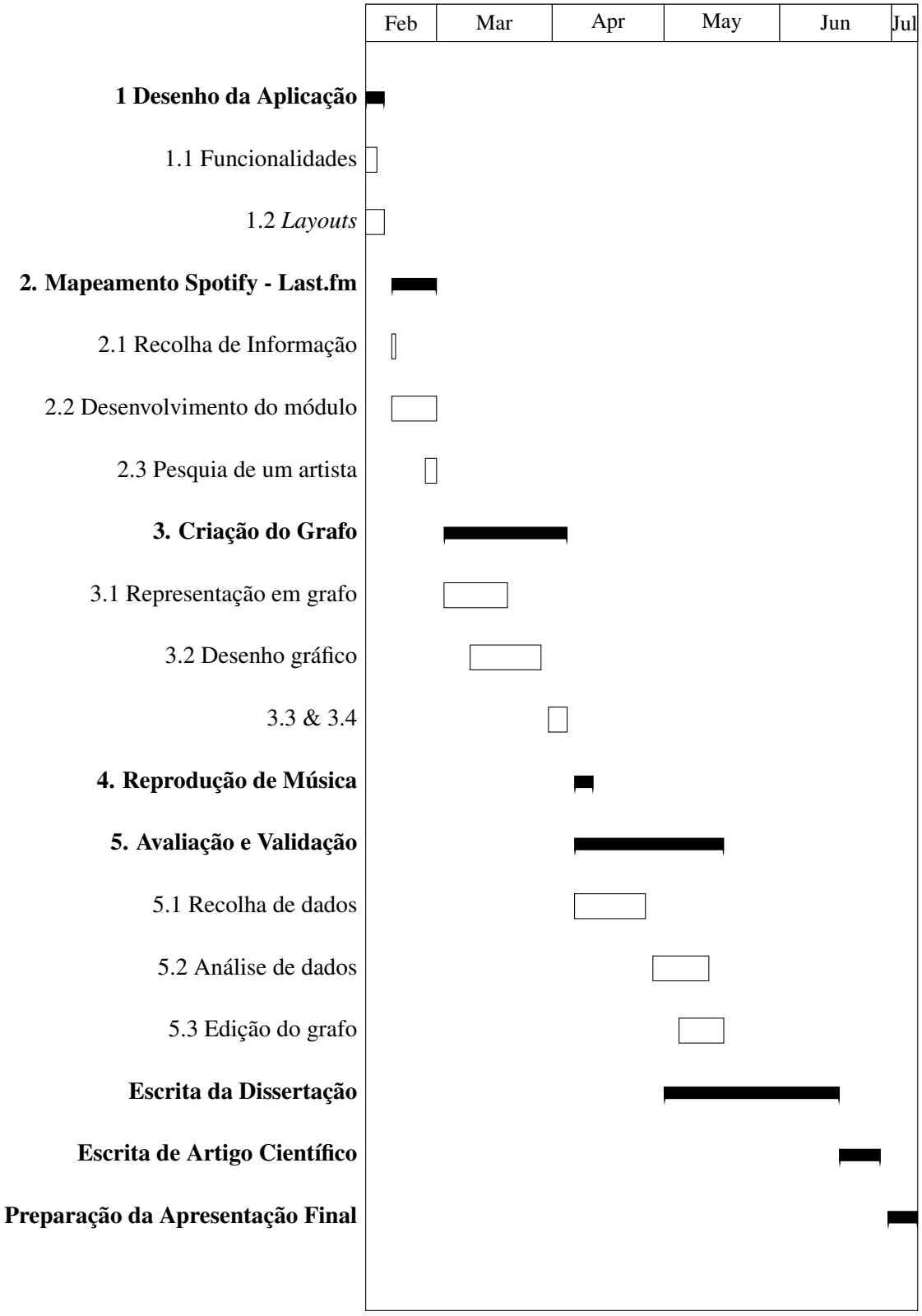



Figura 4.1: Calendarização do Plano de Trabalho

# Capítulo 5

# Conclusões

# Referências


[1] P. Lamere. Creating transparent, steerable recommendations. 2008.

[2] BG Costa, Fabien Gouyon, e L Sarmento. A Prototype for Visualizing Music Artist Networks. 2008. URL: http://www.inescporto.pt/~fgouyon/docs/CostaGouyonSarmento_ARTECH2008.pdf.

[3] L Sarmento e EC Oliveira. Visualizing networks of music artists with rama. *International Conference on Web ...*, 2009. URL: http://repositorio-aberto.up.pt/bitstream/10216/15194/2/18675.pdf.

[4] Diogo Costa, Luis Sarmento, e Fabien Gouyon. RAMA : An Interactive Artist Network Visualization Tool. (i):2, 2009. URL: http://ismir2009.ismir.net/proceedings/LBD-2.pdf.

[5] Fabien Gouyon, Nuno Cruz, e Luis Sarmento. A last.fm and youtube mash-up for music browsing and playlist edition. 2011.