

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Spotify-ed: Music Recommendation and Discovery in Spotify

José Lage Bateira



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Fabien Gouyon

Co-Supervisor: Matthew Davies

June 23, 2014

Spotify-ed: Music Recommendation and Discovery in Spotify

José Lage Bateira

Mestrado Integrado em Engenharia Informática e Computação

June 23, 2014

Abstract

Not so long ago, before the Internet boom, listening or discovering new music was a challenge on its own. Now, with a few clicks one can have on their hands such a vast music catalogue that a human mind cannot compute it.

There are dozens of online services that offer exactly that. Some focus on creation/generation of playlists, others try to expand their music catalogue even further, but others focus more on personalized music recommendation. And these ones present their results to the user with a list or a grid of music artists, for example.

However, lists or grids do not give the user enough information about the relation between the results. One could even say that they are not related to each other, which is not true.

The relations exist and can be represented as a network of interconnected artists in a graph, where a node is a music artist, and each edge between them represents a strong connection. This is the concept that RAMA (Relational Artist MAPs), a project developed at INESC Porto, uses.

From a single search, RAMA is able to draw a graph that helps the user to explore new music that might caught his/her interest in a much more natural way.

Nonetheless, when a user wants to listen to an artist's music, Youtube's stream is used. Although one can find a large catalogue of music in Youtube, this service is not Music Oriented and the sound quality is not adequate for a music streaming service.

Youtube's stream needs to be replaced, and Spotify can provide a quality stream and an accurate music catalogue.

But how can RAMA and Spotify be integrated?

This thesis proposes a Spotify App. Will a Spotify user experience a more pleasant and natural way of music discovery from this graphical representation of artist relations within Spotify, than its standard discovery more (with grids)?

That is the main question that this dissertation urges to answer.

Resumo

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. 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 (Relational Artist MAPs), projeto desenvolvido no INESC Porto, usa.

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.

No entanto, quando um utilizador pretende ouvir uma música de um artista, é usado *stream* do Youtube. Apesar de este oferecer um catálogo alargado de música, o mesmo não é indicado para esta funcionalidade pois não fornece uma API 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 é um deles. Fornece API orientada a música, 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?

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.

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation and Goals	1
1.3	Project	2
1.4	Dissertation Structure	3
2	State of The Art	5
2.1	Introduction	5
2.2	Related and Similar Services	5
2.2.1	Liveplasma - liveplasma.com	5
2.2.2	Tuneglue - audiomap.tuneglue.net	7
2.2.3	MusicRoamer - musicroamer.com	8
2.3	Conclusions	9
3	Projeto	13
3.1	Spotify	14
3.1.1	Ferramentas de Desenvolvimento	14
3.1.2	Experimentações Feitas	20
3.1.3	Conclusão	23
3.2	Tecnologias	23
3.2.1	<i>Spotify Desktop Client</i>	23
3.2.2	Webkit Development Tools - webkit.org	24
3.2.3	Npmjs - npmjs.org	24
3.2.4	Gruntjs - gruntjs.com	27
3.2.5	Arborjs - arborjs.org	27
3.3	Resumo e Conclusões	27
4	Plano de Trabalho	29
4.1	Fases do Projeto	29
4.1.1	Fase 1 - Desenho da Aplicação	29
4.1.2	Fase 2 - Mapeamento de Metadados Spotify em Last.fm	30
4.1.3	Fase 3 - Criação e Edição do grafo	30
4.1.4	Fase 4 - Reprodução de Música	31
4.1.5	Fase 5 - Avaliação e Validação	31
4.2	Calendarização	32
4.3	Resumo	32
5	Conclusões	35

CONTENTS

List of Figures

2.1	liveplasma: search result for "Amália Rodrigues"; upper left corner: artist albums; lower left corner: youtube's <i>mini-player</i>	6
2.2	liveplasma: interface to start playing tracks. <i>Similar</i> button plays tracks from similar artists, whereas, the <i>only</i> button only plays tracks from the specified artist.	6
2.3	Tuneglue: menu que aparece ao clicar num nó.	7
2.4	Tuneglue: grafo depois do primeiro nó ser expandido.	8
2.5	MusicRoamer: Várias opções de pesquisa. Por artista; por <i>keyword</i> e pelo perfil de utilizador do Last.fm	9
2.6	MusicRoamer: Parâmetros de personalização do grafo	9
2.7	MusicRoamer: Representação visual do grafo de artistas	10
2.8	MusicRoamer: Grafo depois de expandir um nó	11
3.1	Spotify: interface do modo de descoberta do <i>desktop client</i>	15
3.2	Spotify: Aplicação Last.fm aberta no <i>Spotify Player</i>	16
3.3	Spotify: <i>Play Button</i> pode ser embebido em <i>websites</i>	17
3.4	Spotify: <i>Follow Button</i> permite seguir um artista.	17
3.5	Experiência com <i>Metadata API</i> e <i>Play Button Widget</i> (código fonte: github.com/carsy/spotify-playground)	21
3.6	<i>Website</i> do RAMA embebido numa Aplicação Spotify	22
3.7	Resultado do teste do elemento <i>canvas</i>	22
3.8	Menu <i>Develop</i>	24
3.9	Webkit: Vista da tab <i>Inspector</i> . Outras ferramentas disponíveis (tabs): <i>Resources</i> , <i>Network</i> , <i>Sources</i> , <i>Timeline</i> , <i>Profiles</i> , <i>Audits</i> e <i>Console</i>	25
3.10	Webkit Network	25
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 <i>JQuery</i> que ocupa 12.72% do tempo de processamento, o que pode indicar um possível ponto de melhoria de performance.	26
3.12	Webkit Audit: 96% do código <i>CSS</i> não está a ser usado, sendo por isso, um ponto de melhoria reduzir a quantidade de informação descarregada.	26
3.13	Webkit Console: Erros de <i>Javascript</i> aparecem destacados para chamar a atenção.	26
4.1	Calendarização do Plano de Trabalho	33

LIST OF FIGURES

Chapter 1

Introduction

1.1 Context

Not so long ago, before the Internet boom, listening or discovering new music was a challenge on its own. Now, with a few clicks one can have on their hands such a vast music catalogue that a human mind is not able to compute.

There is an uncountable number of online services that offer exactly that. Some focus on creation/generation of playlists, others try to expand their music catalogue even further, while others focus more on personalized music recommendations. Most of these, present their results to the user with a list or a grid of music artists, for example.

However, lists or grids do not give the user enough information about the relation between the results [1]. One could even say that they are not related to each other, which is not true.

The relations exist and can be represented as a network of interconnected artists in a graph, where a node is a music artist, and each edge between them represents a strong connection. This is the concept that RAMA (Relational Artist MAPs), a project developed at INESC Porto, uses. [2] [3] [4] [5]

1.2 Motivation and Goals

From a single search, RAMA is able to draw a graph that helps the user to explore new music that might caught his/her interest in a much more natural way.

Nonetheless, when a user wants to listen to an artist's music, Youtube's stream is used. Although one can find a large catalogue of music in Youtube, this service is not music oriented and the sound quality is not adequate for a music streaming service.

Youtube's stream needs to be replaced, and Spotify¹ can provide a quality stream and an accurate music catalogue.

¹<http://spotify.com>

But how can RAMA and Spotify be integrated?

Several possibilities were analysed.

Spotify Play Button²

A Spotify widget that can be embedded in RAMA.

Integrate Spotify User's profile data in RAMA

To help complement artist recommendations.

Spotify App³

A plugin to Spotify's desktop client

This dissertation proposes a Spotify App. Will a Spotify user experience a more pleasant and natural way of music discovery from this graphical representation of artist relations within Spotify, than its standard discovery mode (with lists)?

That is the main question that this dissertation urges to answer.

Moreover, to evaluate and validate the final product, end-user testing will be done to compare Spotify's user experience with and without the developed application.

1.3 Project

The app is meant to be an extra mode for discovering new music in Spotify.

This way, a visual representation of an artist network with a graph, similar to RAMA, is proposed.

The application runs inside the Spotify environment (Spotify's Desktop Client) where its main features are: visualization of relations between artists, starting by the current playing artist; ability to grow the graph by expanding nodes; visualize tags (that describe an artist) in the graph representation.

The tools used in the development of the application were:

Spotify Desktop Client

The developed application is integrated in the Spotify's desktop client.

Webkit Development Tools - webkit.org

This is the engine used to run Spotify Applications.

Npmjs - npmjs.org

Package manager for development dependencies.

Bower - bower.io

Package manager for runtime dependencies.

Gruntjs - gruntjs.com

Manager for automating tasks. Very usefull for tests, code optimization and other repetitive tasks.

Vis.js - visjs.org

Visualization framework.

1.4 Dissertation Structure

This dissertation contains four additional chapters.

In chapter [2](#), related works will be presented to evaluate the current state of the art.

In the [third](#) chapter, the project's details will be explained, starting with an introduction to the Spotify Apps' development environment, the role of the technologies used during development and the app's architecture.

In chapter [4](#), the future work will be presented.

Chapter [5](#) concludes this report.

Introduction

Chapter 2

State of The Art

2.1 Introduction

In this chapter, the most relevant web services for this thesis will be analysed.

The proposed methodology focus on how the content is presented and less on what the content is (without discarding its importance). Even so, some projects that focus on the content will be analysed.

The presented projects often use external data bases (like last.fm) to fetch metadata from. This is the preferred way, since those are the most complete sets of information.

2.2 Related and Similar Services

2.2.1 Liveplasma - liveplasma.com

liveplasma.com is a *flash*¹ application that not only it allows to see a graph of music artists, but also of books and movies.

The interaction with the graph is very faulted: no changes to the graph are allowed, and the user can easily make a mistake and perform unwanted actions like redrawing the graph with another artist as the root node.

In 2.1 one can see the search result for "Amália Rodrigues".

On the left side of the application there are some interesting elements: a grid of the artist's albums, a mini-player (stream from Youtube).

In 2.2 the user can have the choice to play tracks *only* from that artist, or play *similar* artists.

2.2.1.1 Pros

This tools, has two interesting aspects to it:

¹<http://get.adobe.com/flashplayer>

State of The Art

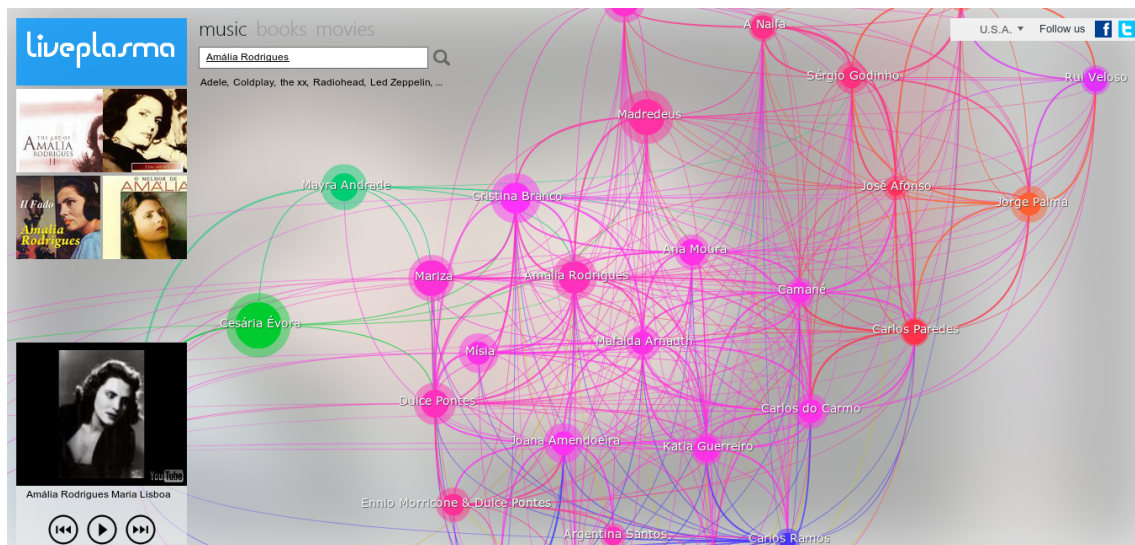


Figure 2.1: liveplasma: search result for "Amália Rodrigues"; upper left corner: artist albums; lower left corner: youtube's *mini-player*

- Links to buy albums of the artist
- Play tracks from similar artists to the search artist.

2.2.1.2 Cons

The graph drawn from this simple search, is very cluttered with edges. Two nodes can have several connections between, which seems to overload the graph and making it very confusing.

Different colours are used, but their meaning remains unknown. One can assume that they represent the similarity between artists, but that is just speculation.

It can also be assumed that the size of the nodes (radius value) can be directly proportional to the artist's popularity, but that is, again, just speculation.

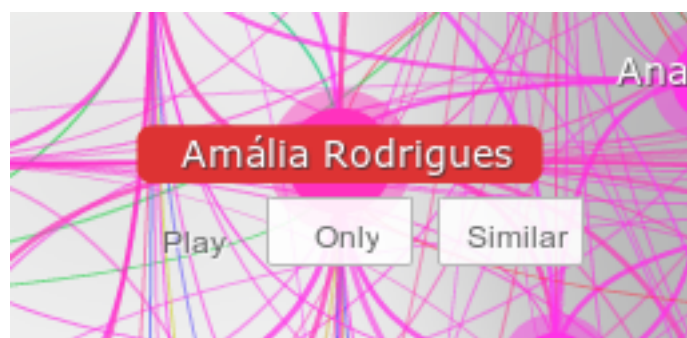


Figure 2.2: liveplasma: interface to start playing tracks. *Similar* button plays tracks from similar artists, whereas, the *only* button only plays tracks from the specified artist.



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

One critical detail is that the user cannot visually point out the search node in the graph, given the lack of visual distinction from the other nodes of the graph 2.1.

2.2.1.3 Summary

In short, liveplasma is not very user friendly. It uses too many colours and edges, which makes the user experience of searching for new music even harder than usual.

2.2.2 Tuneglue - audiomap.tuneglue.net

Tuneglue is another flash application that tries to explore the graphic visualization of network of related artists. Last.fm's metadata API is used to retrieve artist information.

When you start Tuneglue and search for an artist, say "Mariza", the user is presented with a single-node graph. By clicking the node, the user has four options (2.3): expand, releases, lock position and delete.

When you first expand a node, you get the root node with six child nodes 2.4.

So the first feature that brings the user experience to another level (in comparison with liveplasma) is that of graph editing. The user can expand, fix and delete every single node in the graph.

2.2.2.1 Pros

Tuneglue gives control to the user. On one hand, the user is able to craft a graph and tailor it to its needs. The user feels that graph is its own creation.

2.2.2.2 Cons

On the other hand, the user has the responsibility to create the whole graph, which might be too much trouble and deteriorate the user experience.

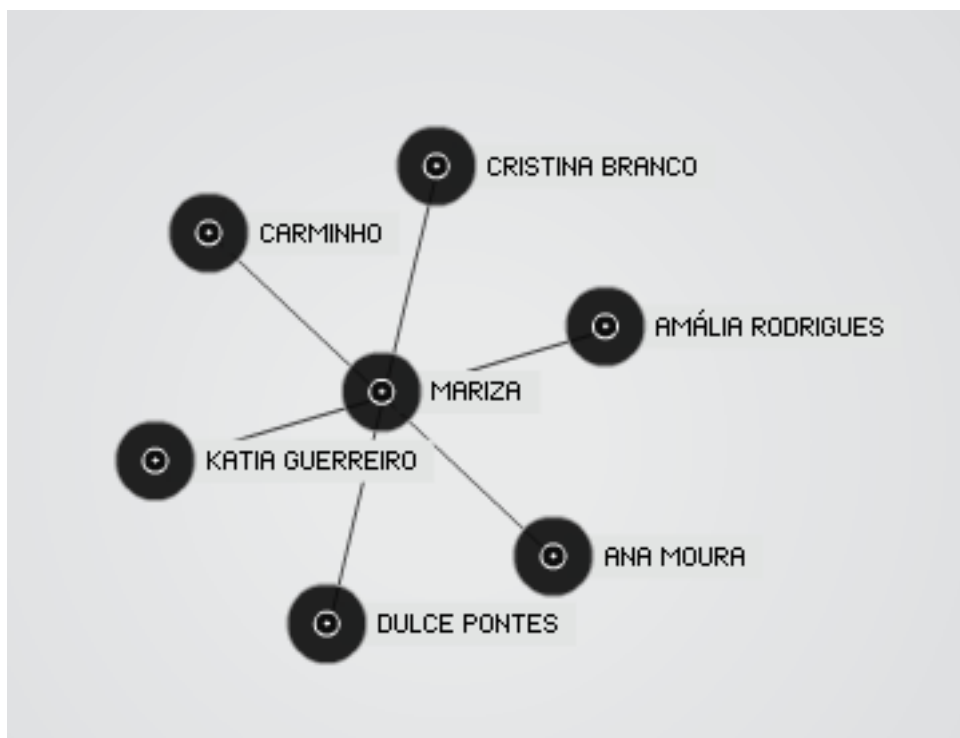


Figure 2.4: Tuneglue: grafo depois do primeiro nó ser expandido.

Again the root node is not highlighted, which might leave the user lost when the graph gets more and more complex.

2.2.2.3 Summary

Tuneglue takes the approach to give the user the power to create what he wants. But with no limit, the user can easily create a very complex graph that deteriorates the user experience.

2.2.3 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.

2.2.3.1 Pros

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

The image shows a horizontal search bar with three input fields and corresponding 'Go' buttons. The first field is labeled 'Artist Name' and is empty. The second field is labeled 'Keyword (dance)' and contains the text 'dance'. The third field is labeled 'Last.FM username' and is empty. Each field has a 'Go' button to its right.

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

Pesquisa por perfil do Last.fm

Esta pesquisa gera um grafo para cada artista (os mais ouvidos pelo utilizador)

Todas estas formas de pesquisa desenharam 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.

2.2.3.2 Cons

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). As linhas começam a se sobrepor e alguns nós ficam pouco legíveis.

2.2.3.3 Summary

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

2.3 Conclusions

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



Figure 2.6: MusicRoamer: Parâmetros de personalização do grafo

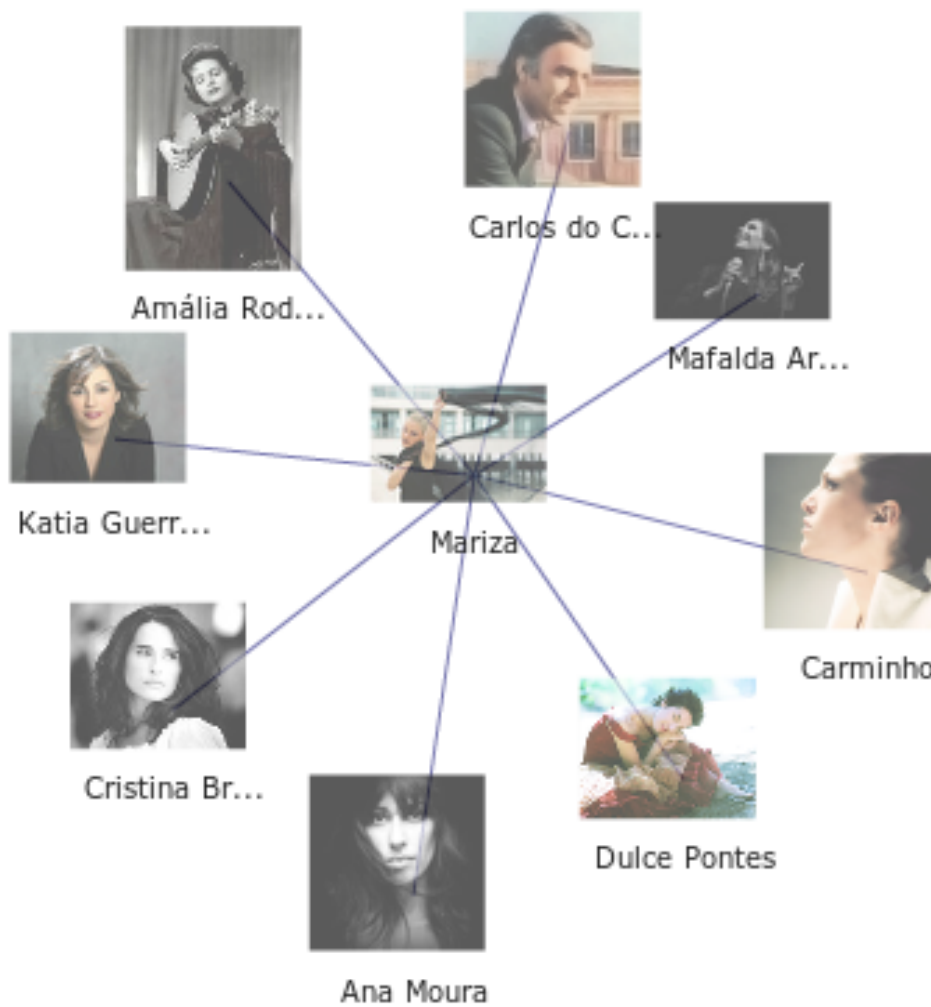


Figure 2.7: MusicRoamer: Representação visual do grafo de artistas

- audiomapa.tuneglue.net
- musicroamer.com
- discovr.info
- ifyoudig.net
- pitchfork.com
- hypem.com
- awdio.com

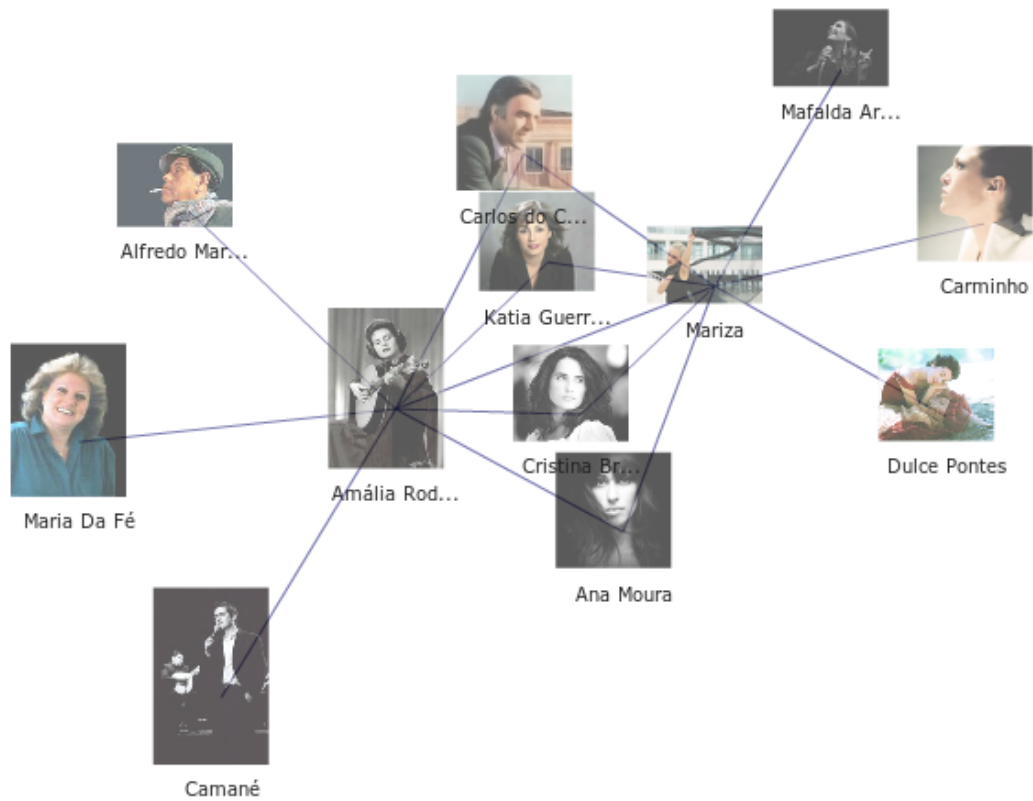


Figure 2.8: MusicRoamer: Grafo depois de expandir um nó

- 8tracks.com
- tastekid.com
- songza.com
- thesixtyone.com
- mog.com
- stereogum.com
- gigfi.com
- jango.com
- soundcloud.com
- grooveshark.com

State of The Art

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, por exemplo, será limitar o fator de ramificação a um máximo que não cause este problema.

Chapter 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 ferramentas¹ 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 Spotify² 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 reproduzidor 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/>

Projeto

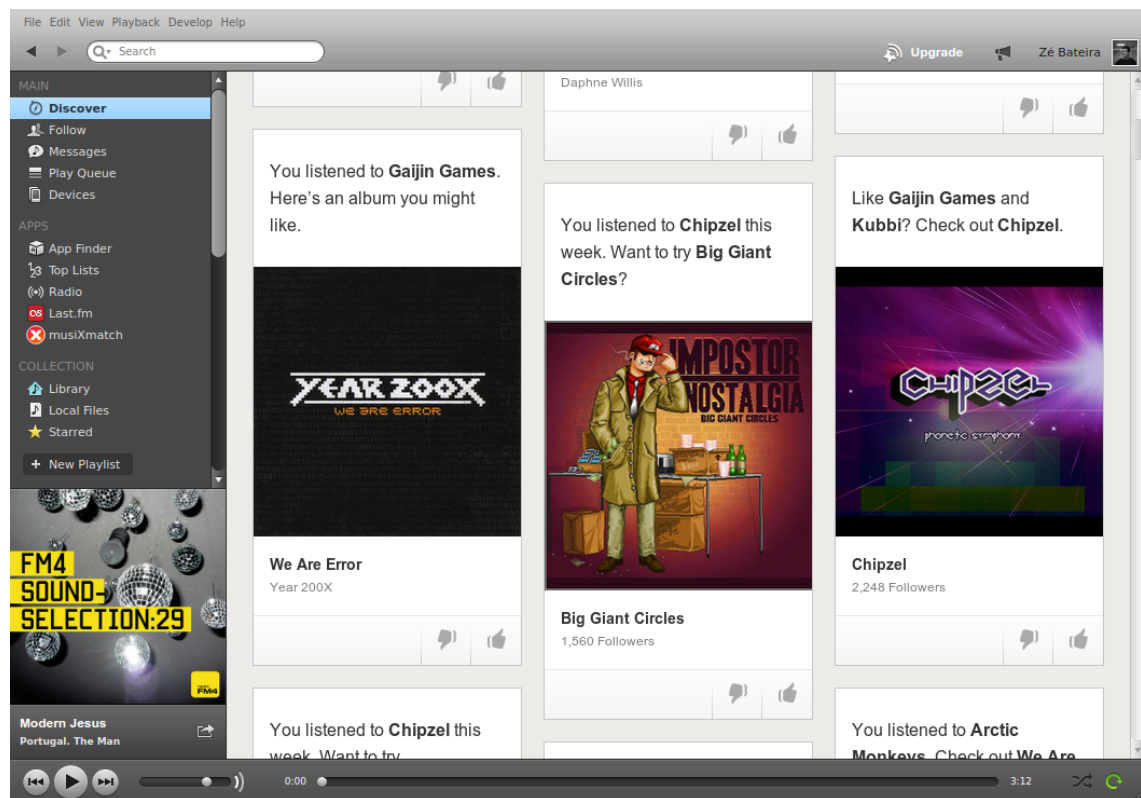


Figure 3.1: Spotify: interface do modo de descoberta do *desktop client*

Projeto

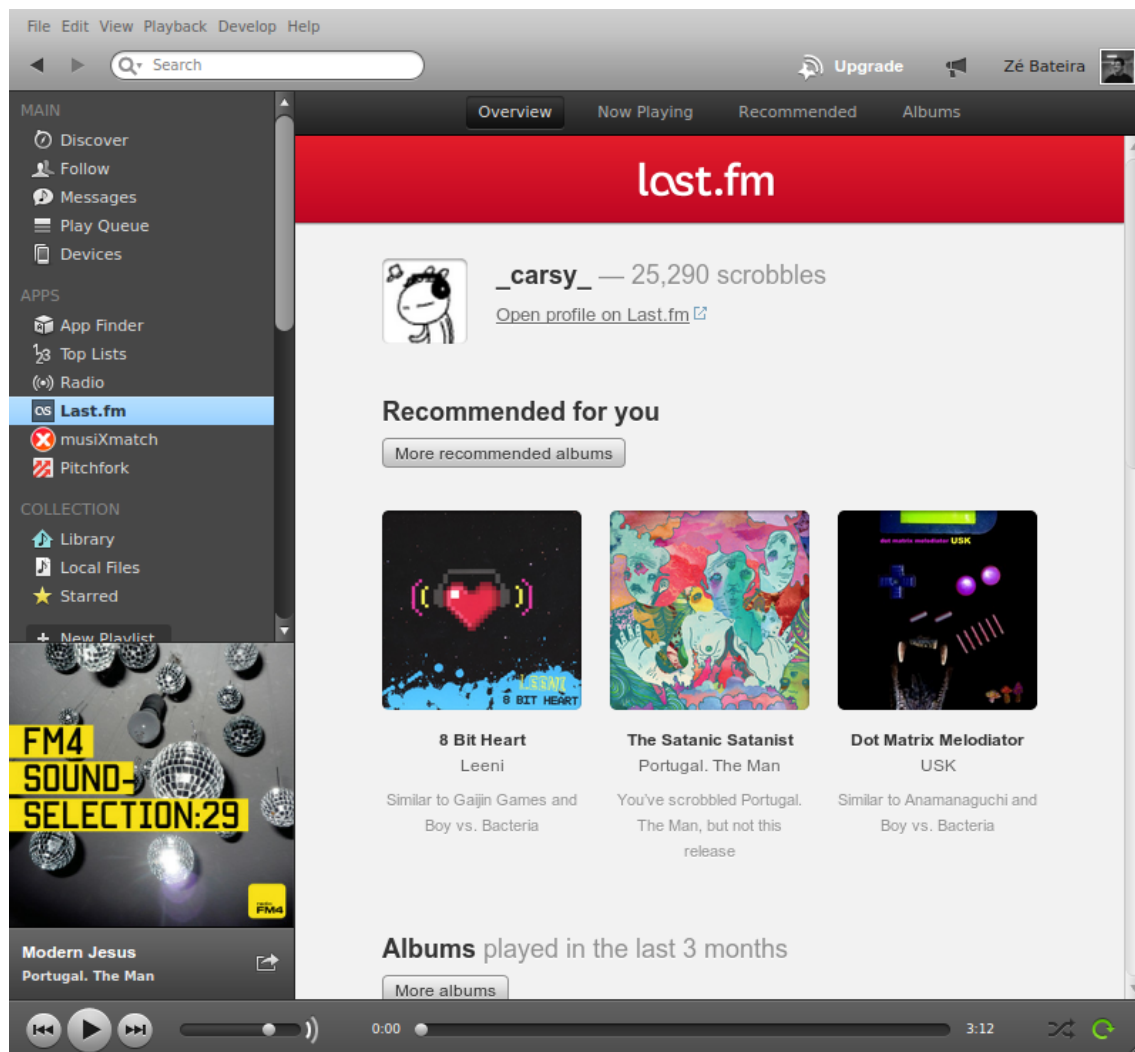


Figure 3.2: Spotify: Aplicação Last.fm aberta no *Spotify Player*

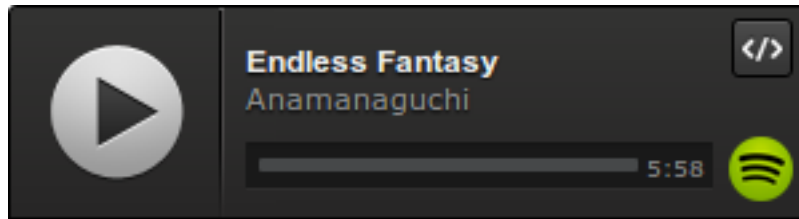


Figure 3.3: Spotify: *Play Button* pode ser embebido em *websites*.

3.1.1.2 Spotify Widgets

As Widgets⁶ 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>



Figure 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 SDK⁷ 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 seu identificador único. Esse identificador é um *URI* da forma:

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

Exemplo:

`spotify:artist:65nZq8l5VZRG4X445F5kmN`, é o identificador único da artista "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 formato *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:

```
1 {
2   "info": {
3     "num_results": 2,
4     "limit": 100,
5     "offset": 0,
6     "query": "camane",
7     "type": "artist",
8     "page": 1
9   },
10  "artists": [
11    {
12      "href": "spotify:artist:3MLPFTe4BrpEV2e0VG0gLK",
13      "name": "Camane",
14      "popularity": "0.27"
15    },
16    {
17      "href": "spotify:artist:5Gwulm1LfURW7dbZD1V3zX",
18      "name": "Sergio Godinho/Camane/Carlos Do Carmo",
19      "popularity": "0"
20    }
21  ]
22 }
```

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:3MLPFTe4BrpEV2e0VG0gLK>

Retorna:

```

1 {
2   "info": {
3     "type": "artist"
4   },
5   "artist": {
6     "href": "spotify:artist:3MLPFTe4BrpEV2e0VG0gLK",
7     "name": "Camane"
8   }
9 }
```

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:

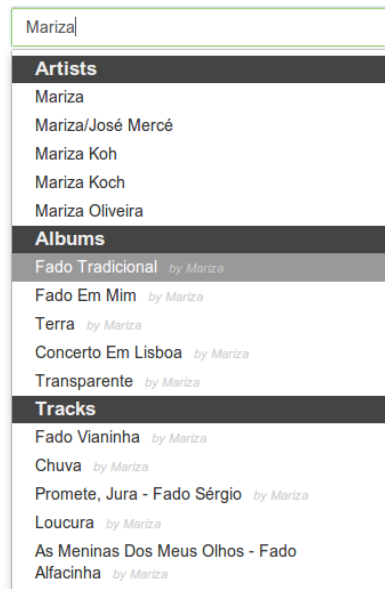
```

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

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

Projeto

Spotify Playground



(a) Resultado da pesquisa "Mariza"

Spotify Playground



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

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

Projeto

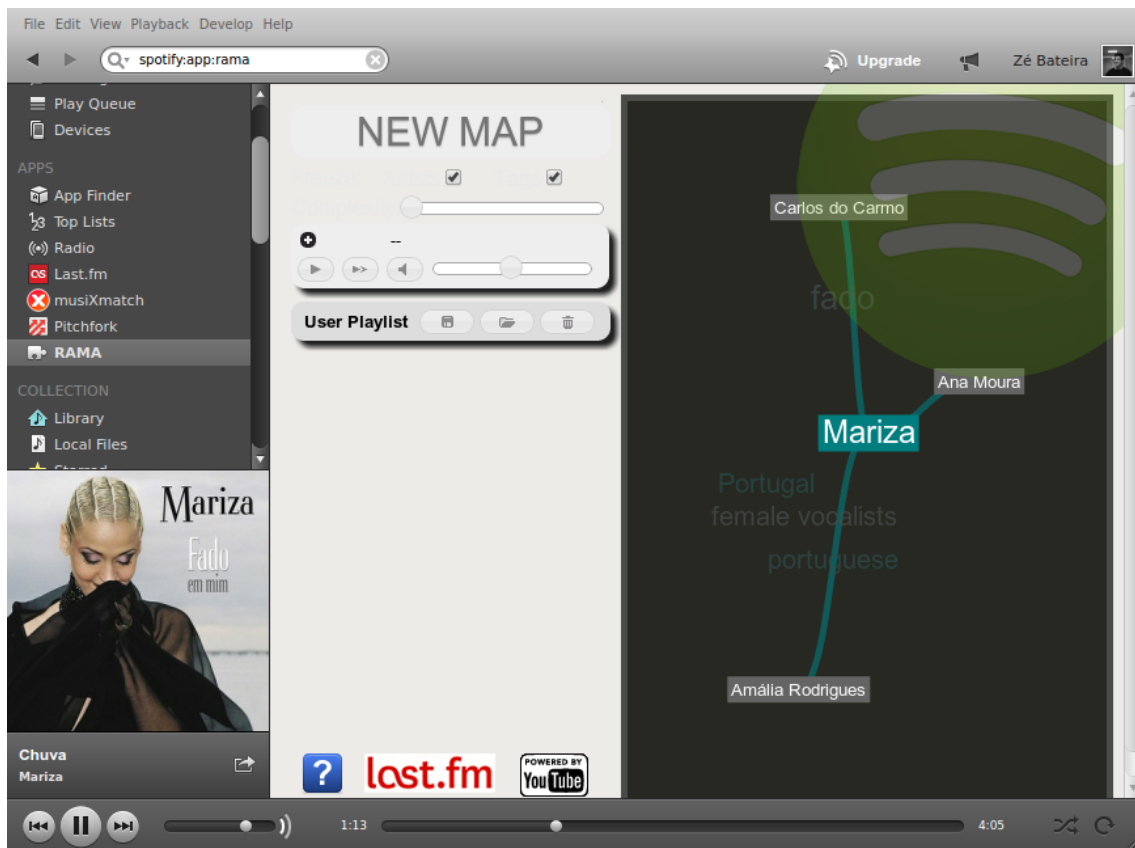


Figure 3.6: Website do RAMA embebido numa Aplicação Spotify

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.

Canvas	20
canvas element ?	Yes ✓
2D context ?	Yes ✓
Text ?	Yes ✓

Figure 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 otimização 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*:

```
1 {
2   "AppName": {
3     "en": "RAMA"
4   },
5   "BundleIdentifier": "rama",
6   "AppDescription": {
7     "en": "RAMA: Relational Artist MApps"
8   },
9   "AcceptedLinkTypes": [
10    "playlist"
11  ],
12   "BundleType": "Application",
13   "BundleVersion": "0.2",
14   "DefaultTabs": [
15     {
16       "arguments": "index",
17       "title": {
18         "en": "Home"
19       }
20     }
21  ],
22   "Dependencies": {
23     "api": "1.10.2",
24     "views": "1.18.1"
25  },
```

¹¹ ficheiro situado na *root* da pasta do projeto

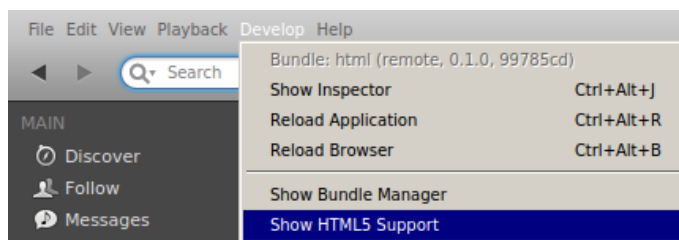


Figure 3.8: Menu *Develop*

```

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:

Projeto

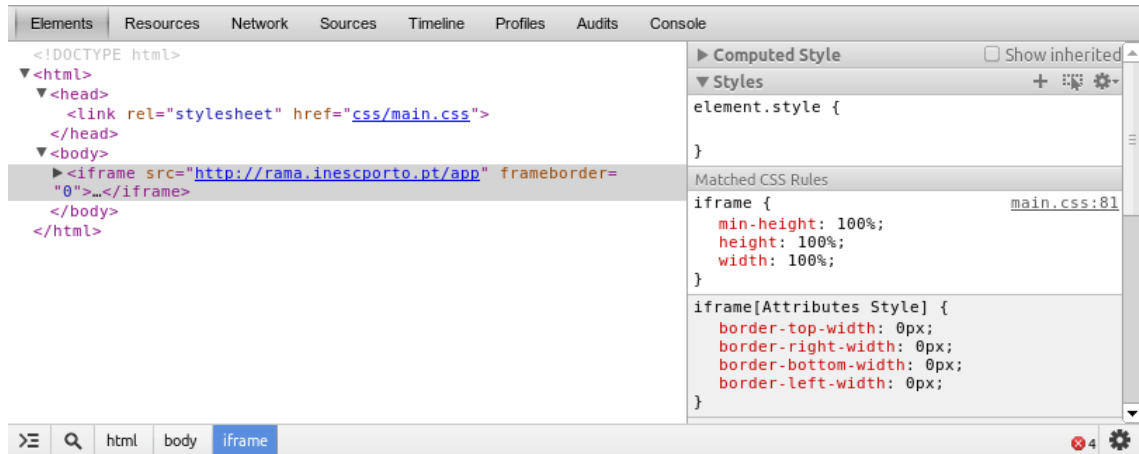


Figure 3.9: Webkit: Vista da tab *Inspector*. Outras ferramentas disponíveis (tabs): *Resources*, *Network*, *Sources*, *Timeline*, *Profiles*, *Audits* e *Console*.

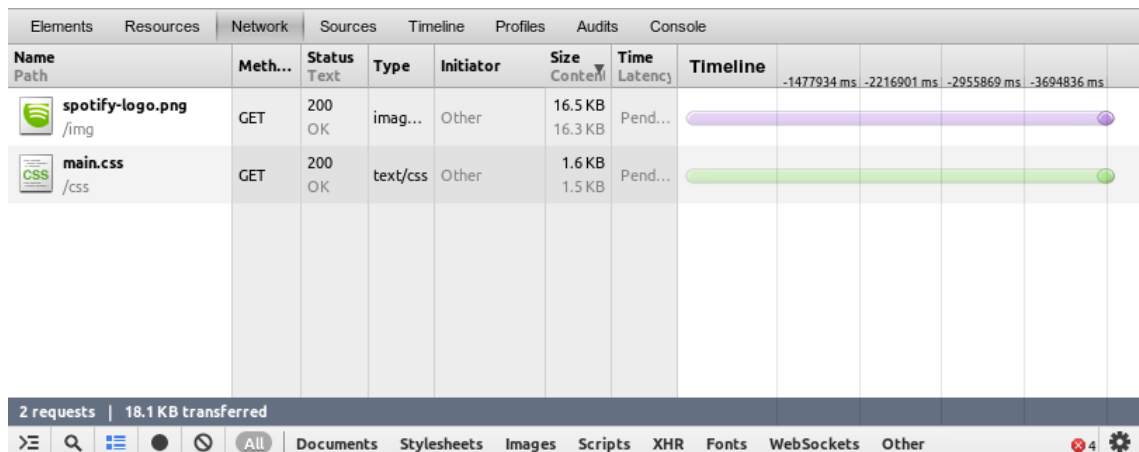


Figure 3.10: Webkit Network

Projeto

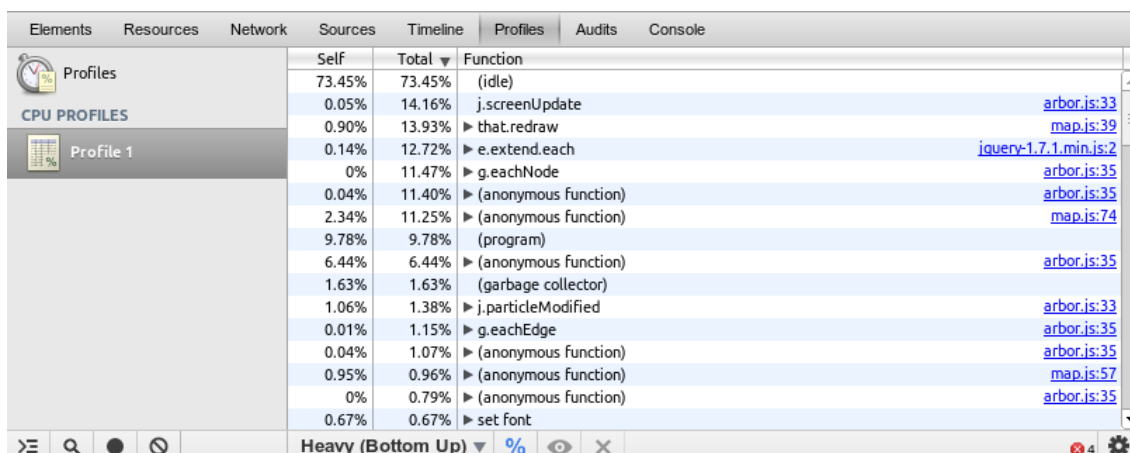


Figure 3.11: Webkit Profile: É possível ver que a renderização do grafo é o que ocupa mais tempo de processamento como esperado. No entanto, existe uma parte de *jQuery* que ocupa 12.72% do tempo de processamento, o que pode indicar um possível ponto de melhoria de performance.

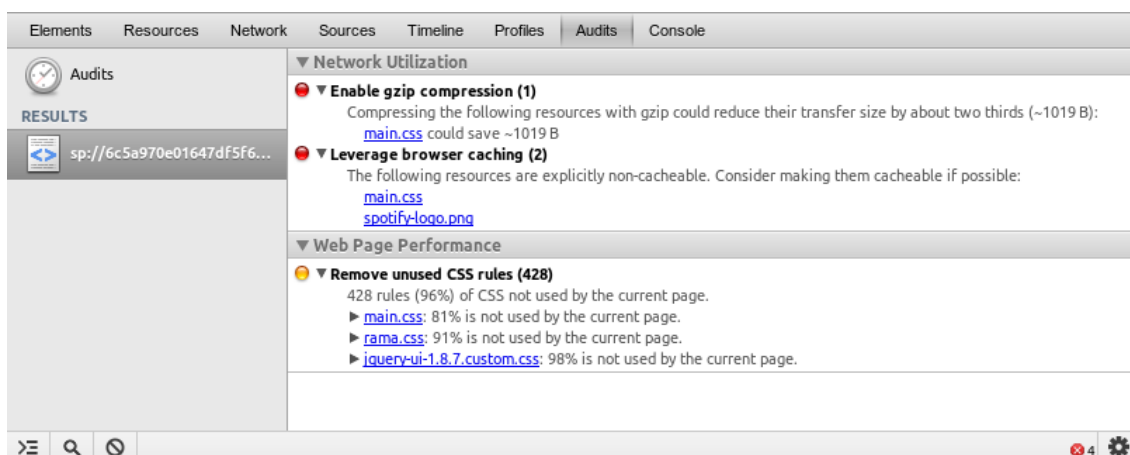


Figure 3.12: Webkit Audit: 96% do código CSS não está a ser usado, sendo por isso, um ponto de melhoria reduzir a quantidade de informação descarregada.

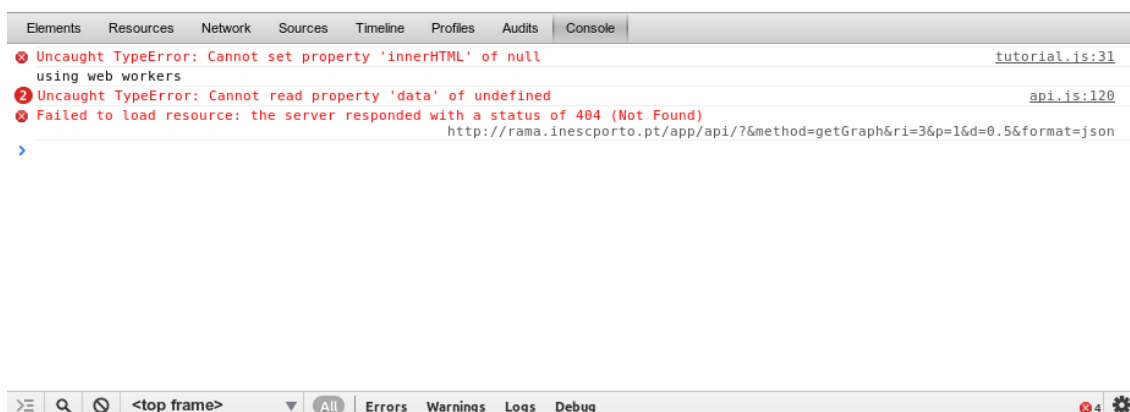


Figure 3.13: Webkit Console: Erros de *Javascript* aparecem destacados para chamar a atenção.

```
1 {  
2   "name": "RAMA",  
3   "devDependencies": {  
4     "grunt": "~0.4.2",  
5     "grunt-contrib-jshint": "*",  
6     "grunt-contrib-jasmine": "*",  
7     "grunt-contrib-watch": "*"  
8   },  
9   "version": "0.1.0"  
10 }
```

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 Resumo e Conclusões

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

Apesar de as outras opções serem também viáveis, a possibilidade de poder integrar uma interface estilo RAMA num ambiente que os utilizadores já se sentem confortáveis (Spotify), é muito favorável a que seja melhor aceite pelos mesmos.

É esperado que as tecnologias a usar ajudem no desenvolvimento desta dissertação.

Em suma, será desenvolvida uma Aplicação Spotify que implemente os módulos 4 e 5.

Projeto

Chapter 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 delas 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 utilizar 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 2D [3.2.5](#). 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 músicas 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 guardar 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.

Plano de Trabalho

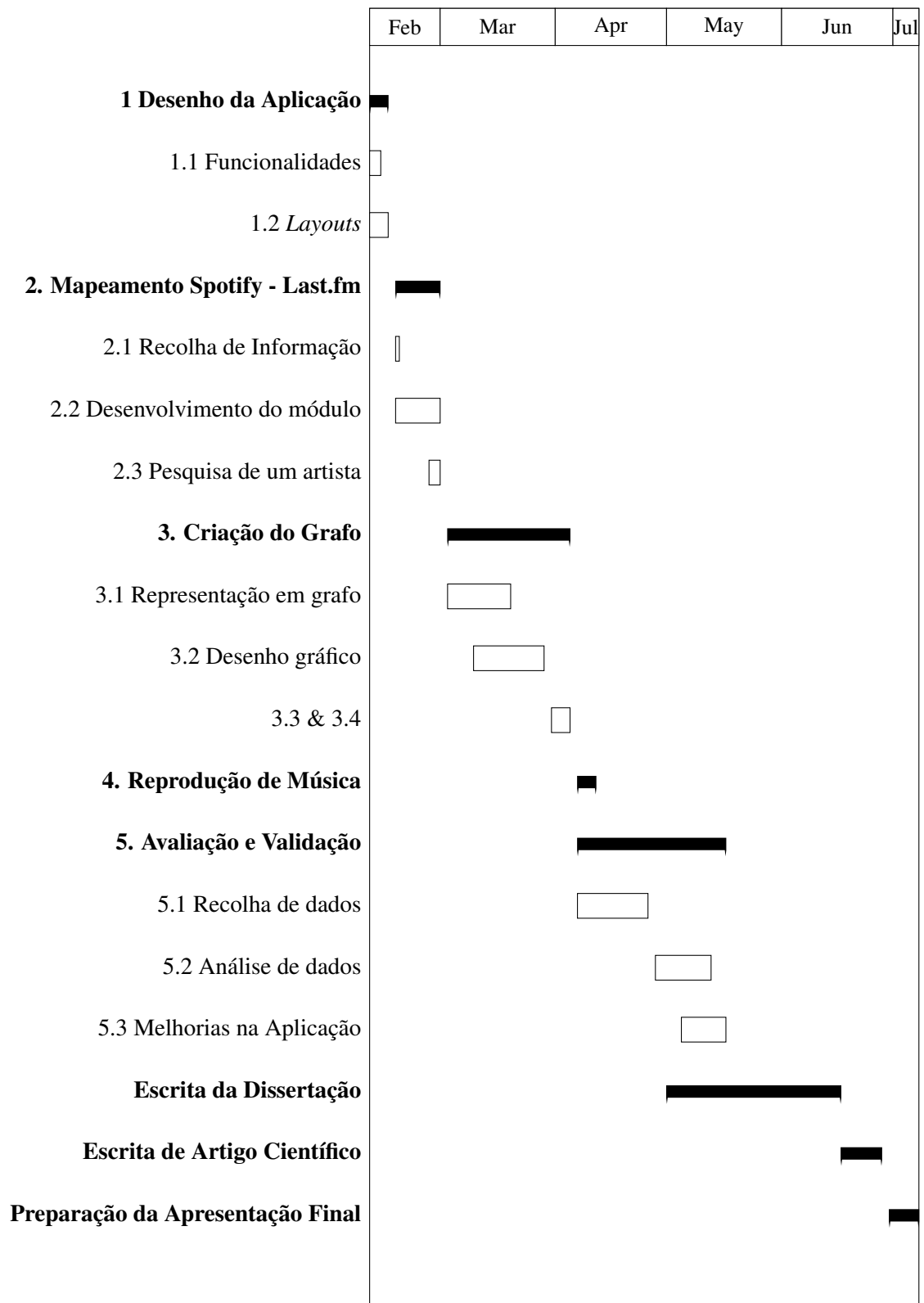


Figure 4.1: Calendarização do Plano de Trabalho

Plano de Trabalho

Chapter 5

Conclusões

Depois de apresentada a revisão bibliográfica, é seguro dizer que o trabalho proposto irá trazer mais respostas à questão da melhoria da experiência musical do utilizador num ambiente de descoberta de nova música. Com o plano apresentado, procura-se concluir que a injeção deste método de descoberta e recomendação de música na experiência dos utilizadores Spotify é um tanto mais vantajoso que os métodos mais comuns.

Conclusões

References

- [1] P. Lamere. Creating transparent, steerable recommendations. 2008.
- [2] BG Costa, Fabien Gouyon, e L Sarmiento. A Prototype for Visualizing Music Artist Networks. 2008. URL: http://www.inescporto.pt/~fgouyon/docs/CostaGouyonSarmiento_ARTECH2008.pdf.
- [3] L Sarmiento 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 Sarmiento, 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 Sarmiento. A last.fm and youtube mash-up for music browsing and playlist edition. 2011.