

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

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	Context and Methodologies	13
3.1	Introducing Spotify	14
3.1.1	Ferramentas de Desenvolvimento	14
3.1.2	Experimentações Feitas	20
3.1.3	Conclusão	23
3.2	Technologies used	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	Conclusions	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: Search options. by artist; by keyword and by Last.fm username	9
2.6	MusicRoamer: Visual representation of the artist graph	10
2.7	MusicRoamer: Personalizable parameters for the graph	11
2.8	MusicRoamer: The graph after expanding one node	12
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 3

Context and Methodologies

The primal objective of this dissertation, as referred in chapter 1, is to develop one or more software modules that will improve Spotify Users' music discovery and recommendation experience using visual tools to represent the music artists' relations and Spotify's high quality music streaming service.

The initial proposal was to develop a module that implements, at least, one of the following modules:

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 Introducing 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/>

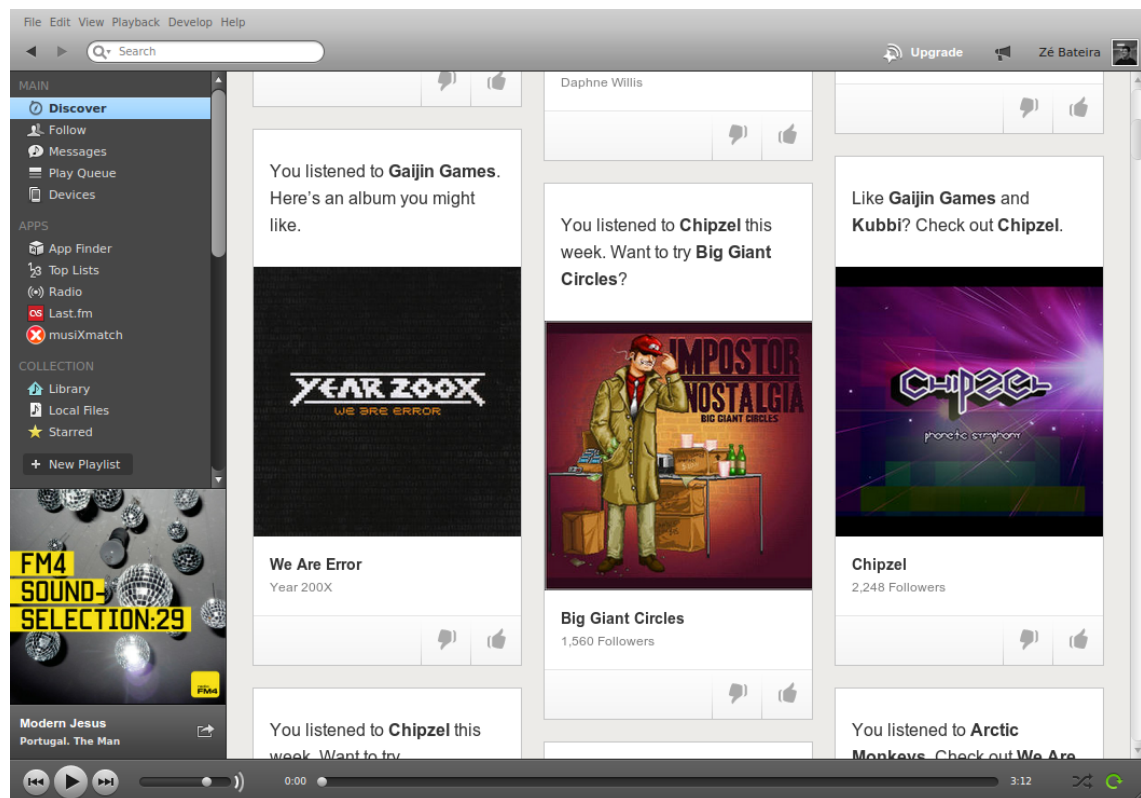


Figure 3.1: Spotify: interface do modo de descoberta do *desktop client*

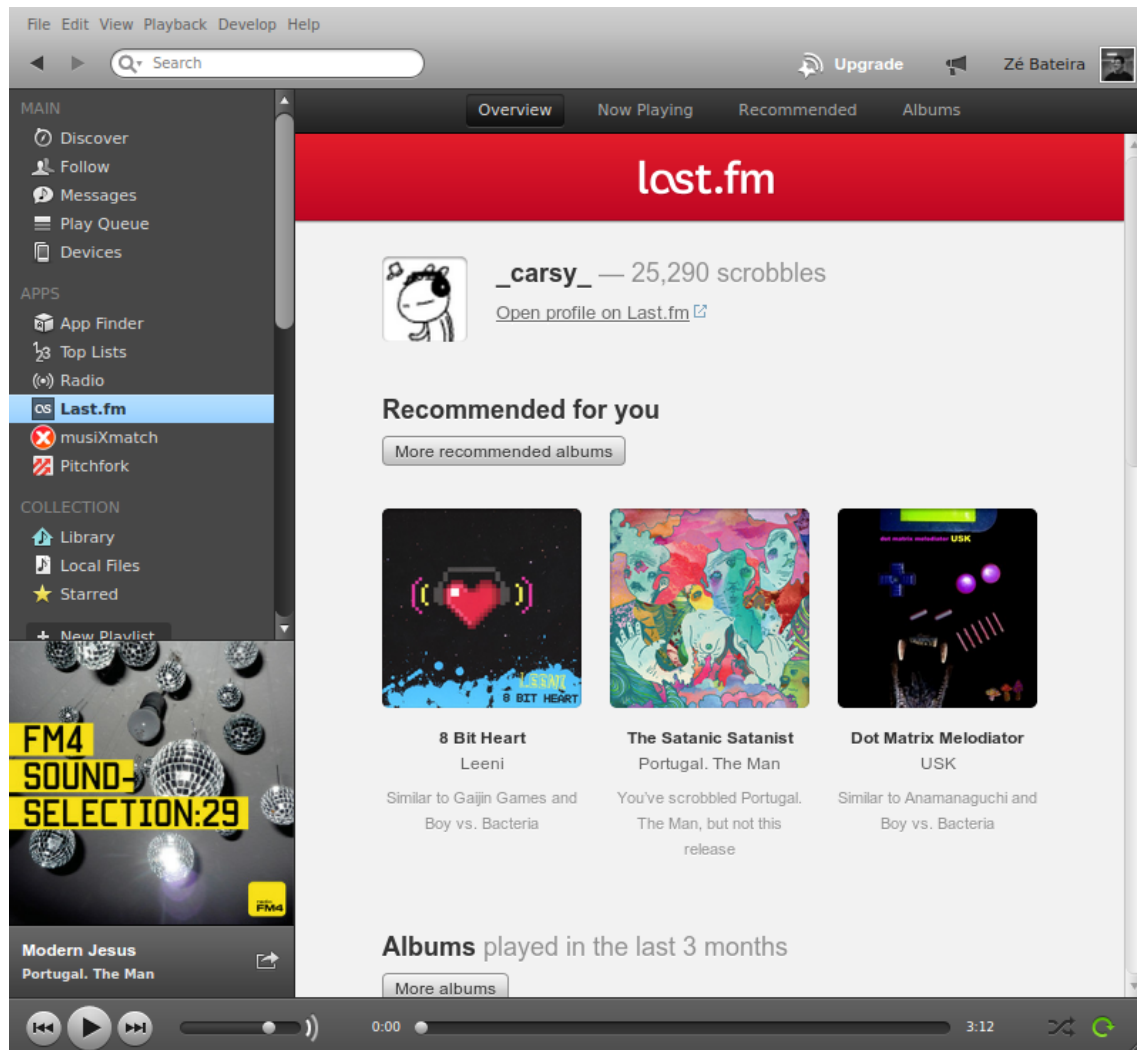


Figure 3.2: Spotify: Aplicação Last.fm aberta no *Spotify Player*

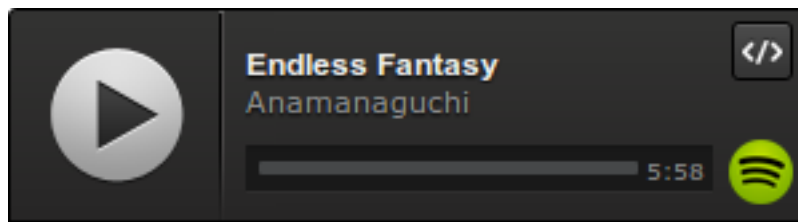


Figure 3.3: Spotify: *Play Button* pode ser embebido em *websites*.

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>

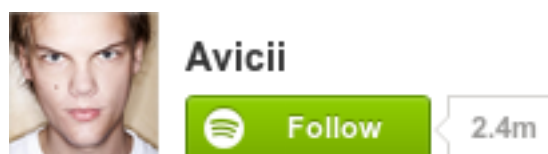


Figure 3.4: Spotify: *Follow Button* permite seguir um artista.

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.

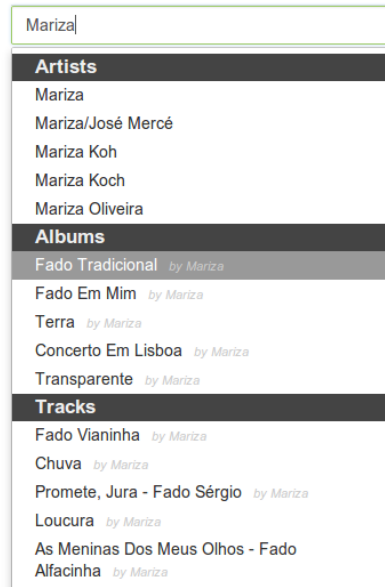
Outra experiência foi realizada para verificar se é possível usar o elemento *canvas* numa Aplicação Spotify. Isto é necessário pois será a única forma de poder desenhar graficamente o grafo. Para isso foi apenas necessário criar uma aplicação com o seguinte código fonte:

```

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

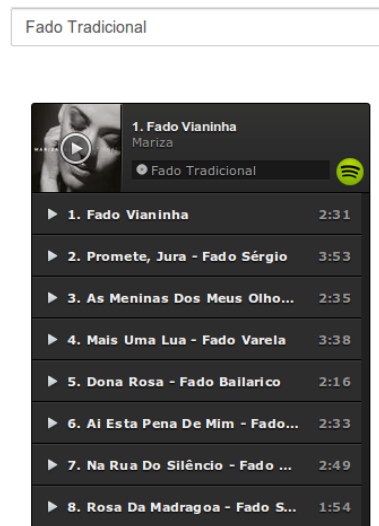
Listing 3.4: Elemento *iframe* que embebe o *website* do RAMA na aplicação

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)

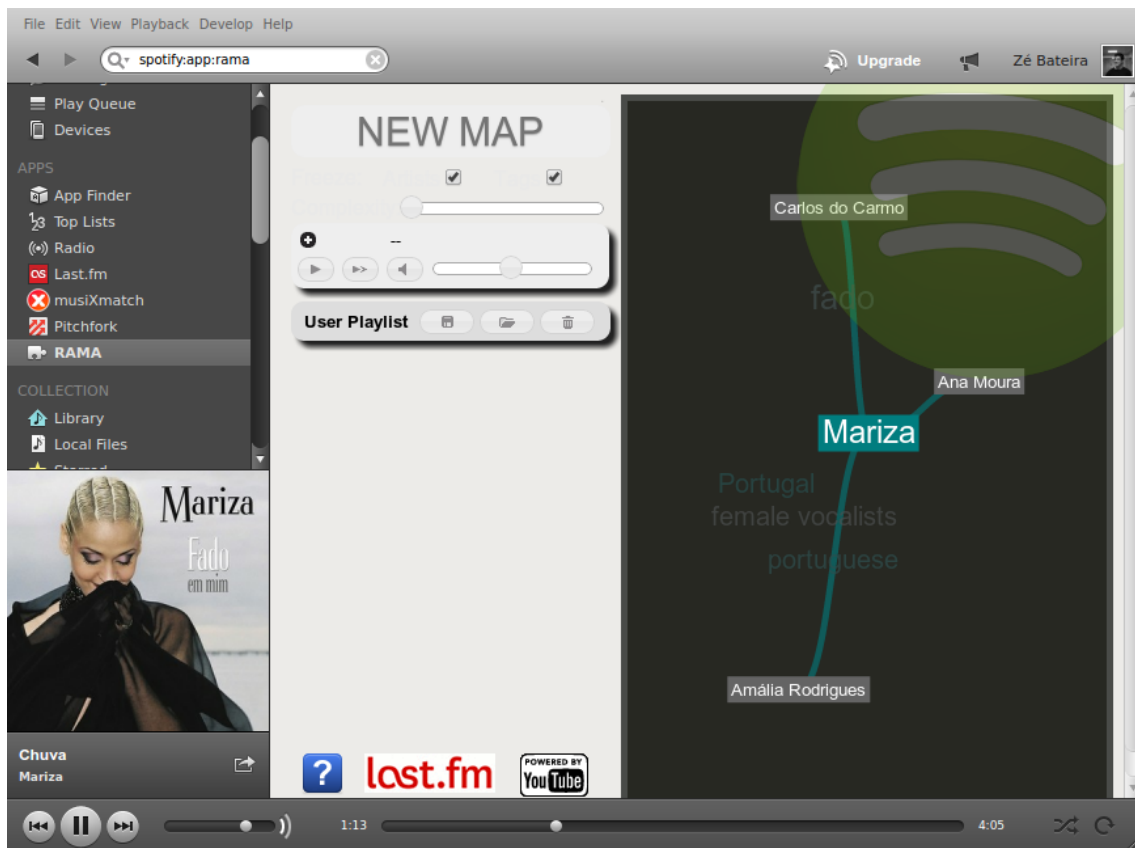


Figure 3.6: Website do RAMA embebido numa Aplicação Spotify

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 Technologies used

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.

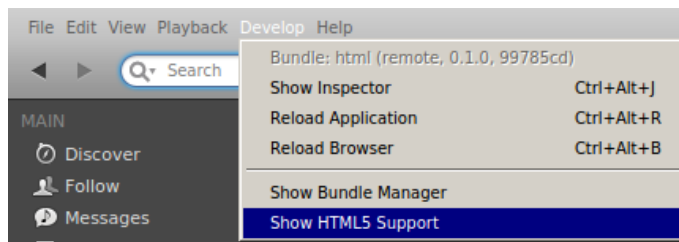
Exemplo de ficheiro *manifest.json*:

```

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:

Context and Methodologies

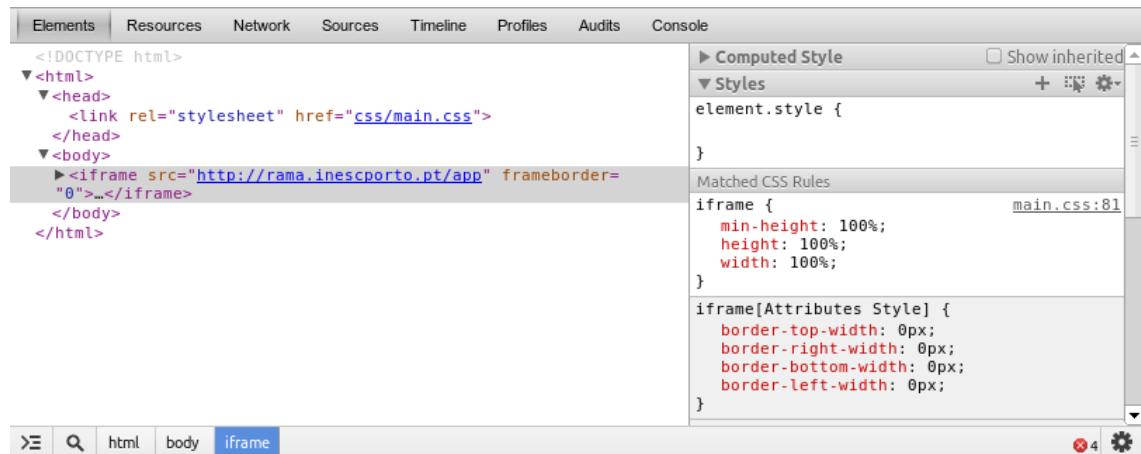


Figure 3.9: Webkit: Vista da tab *Inspector*. Outras ferramentas disponíveis (tabs): *Resources*, *Network*, *Sources*, *Timeline*, *Profiles*, *Audits* e *Console*.

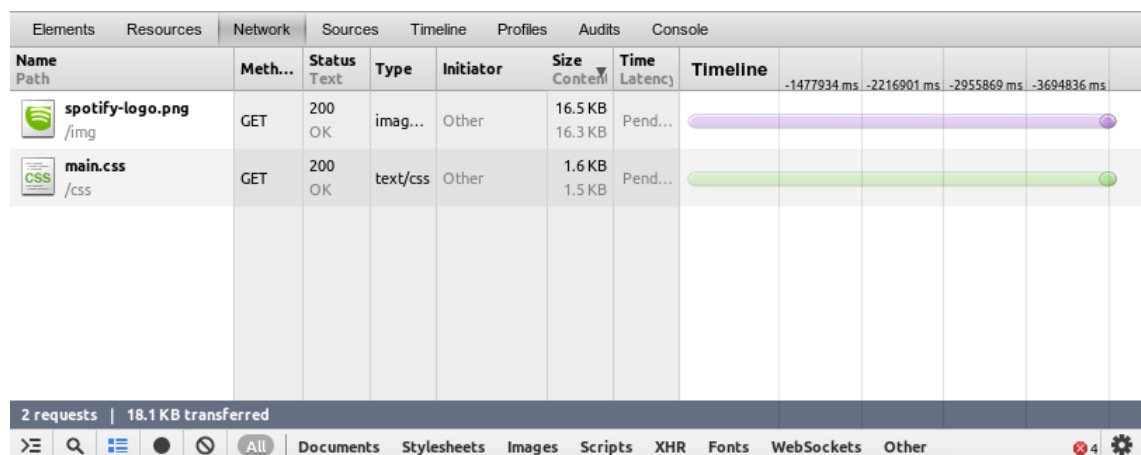


Figure 3.10: Webkit Network

Context and Methodologies

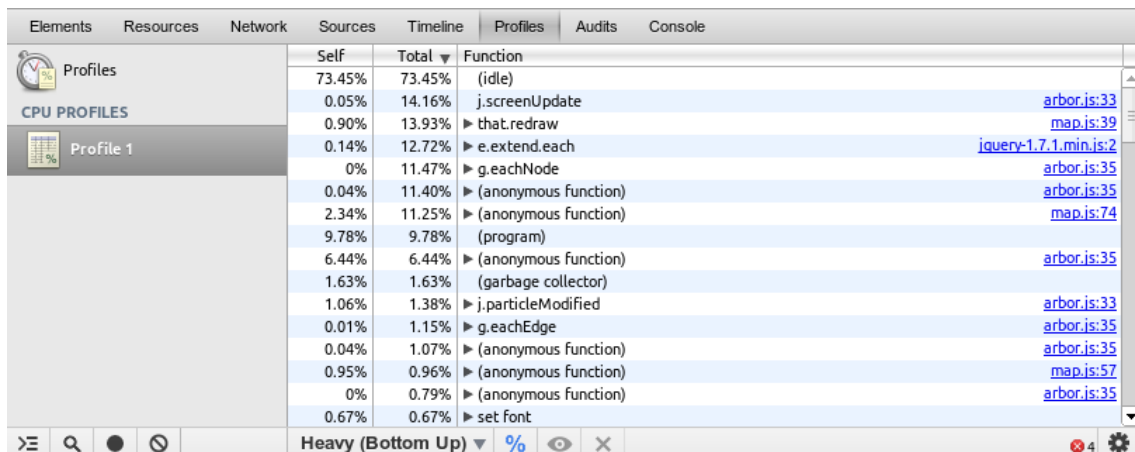


Figure 3.11: Webkit Profile: É possível ver que a renderização do grafo é o que ocupa mais tempo de processamento como esperado. No entanto, existe uma parte de *jQuery* que ocupa 12.72% do tempo de processamento, o que pode indicar um possível ponto de melhoria de performance.

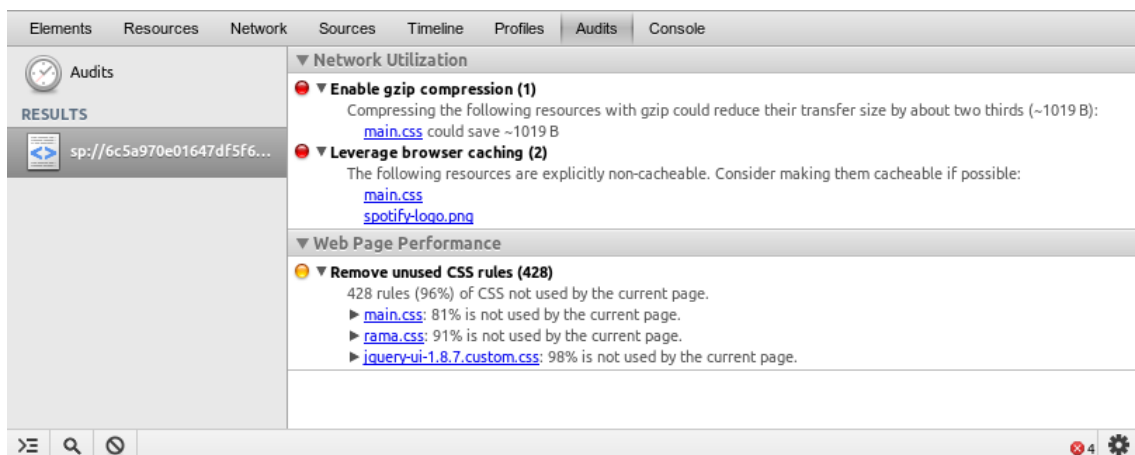


Figure 3.12: Webkit Audit: 96% do código CSS não está a ser usado, sendo por isso, um ponto de melhoria reduzir a quantidade de informação descarregada.

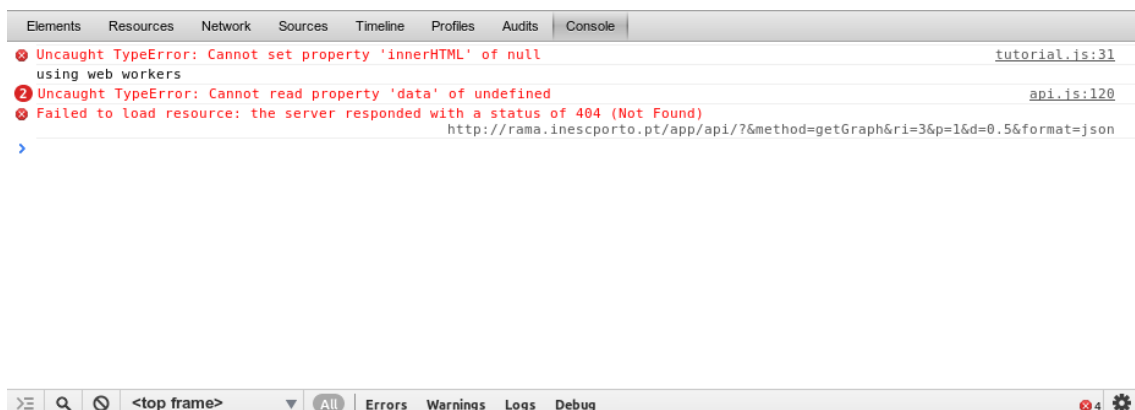


Figure 3.13: Webkit Console: Erros de *Javascript* aparecem destacados para chamar a atenção.

```
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 Conclusions

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.

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.