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

CONTENTS

List of Figures

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 streaming service to provide high quality music stream.

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

1. Integrate Spotify's music stream into RAMA's website
2. Integrate information from the Spotify user into RAMA
3. Improve RAMA's features and design
4. Integrate the RAMA concept into a Spotify Application
5. Integrate RAMA's playlist generation into a Spotify Application
6. Integrate some of the above mentioned modules into a Mobile Application

The first three functionalities (1, 2 and 3) focus on improving RAMA using Spotify's API, i.e. to integrate Spotify into RAMA. Whereas 4 and 5 aim to integrate RAMA's concept into Spotify, through a Spotify Application (it would work as a plugin to Spotify's Desktop Client). The last one (6) would focus on implementing the previous functionalities into an Android, iOS or Windows Phone Application.

This chapter aims to analyse every single drawback of each possibility that affects the choice of which modules do develop, and on which environments it fits better: Spotify Application, Mobile Application, or RAMA improvements.

At first, Spotify's development environment will be introduced 3.1 in order to assess which tools are available for developers. Next, the available tools will be evaluated in order to determine which ones fit the proposed modules to be developed, mostly, through experiments.

By the end of this chapter the modules to be developed should be clearly stated, as well as which tools will be used in the prototype.

The prototype should pursue the objective of contributing to an improved user experience when discovering new music taking advantage of visual tools that implement RAMA's concept.

3.1 Introducing Spotify

Spotify is a Music Streaming Service that allows, through an Internet connection, the user to listen to any track (if available in the user's country) in Spotify's catalogue. The service was launched in 2008 with a native desktop client application.

Now, the service has several types of clients available to the users: desktop client, webplayer and mobile applications.

Desktop Client Desktop version of Spotify, with Windows and Mac versions (and also a Linux preview version).

Webplayer Web version of Spotify. This was released in 2013, and spotify still advises the use of the native applications for a better user experience.

Mobile Applications The mobile applications are available for Android and iOS devices.

3.1.1 Development Tools

Spotify provides a set of tools[1] to develop third-party applications (websites, native applications and mobile applications) and Spotify Applications (that run inside Spotify's Desktop Client). There are five tools, each with different purposes.

3.1.1.1 Spotify Apps

Spotify Applications[2] are a special case in the whole set of tools provided by Spotify. These applications are designed to run *inside* the Desktop Client. Hence, its development is also inside the same environment.

Spotify users can run and install applications from the store called "App Finder". All the applications are free.

In 3.1 on can see the interface of the desktop client. In this case, the discovery mode's interface.

On the left side, in the menu, bellow the "App Finder" item, appears all the applications the user as installed from the store.

In 3.2 the official Last.fm application is opened. Note how the space filled by the applications are always the same.

[1] http://developer.spotify.com/technologies
[2] https://developer.spotify.com/technologies/apps

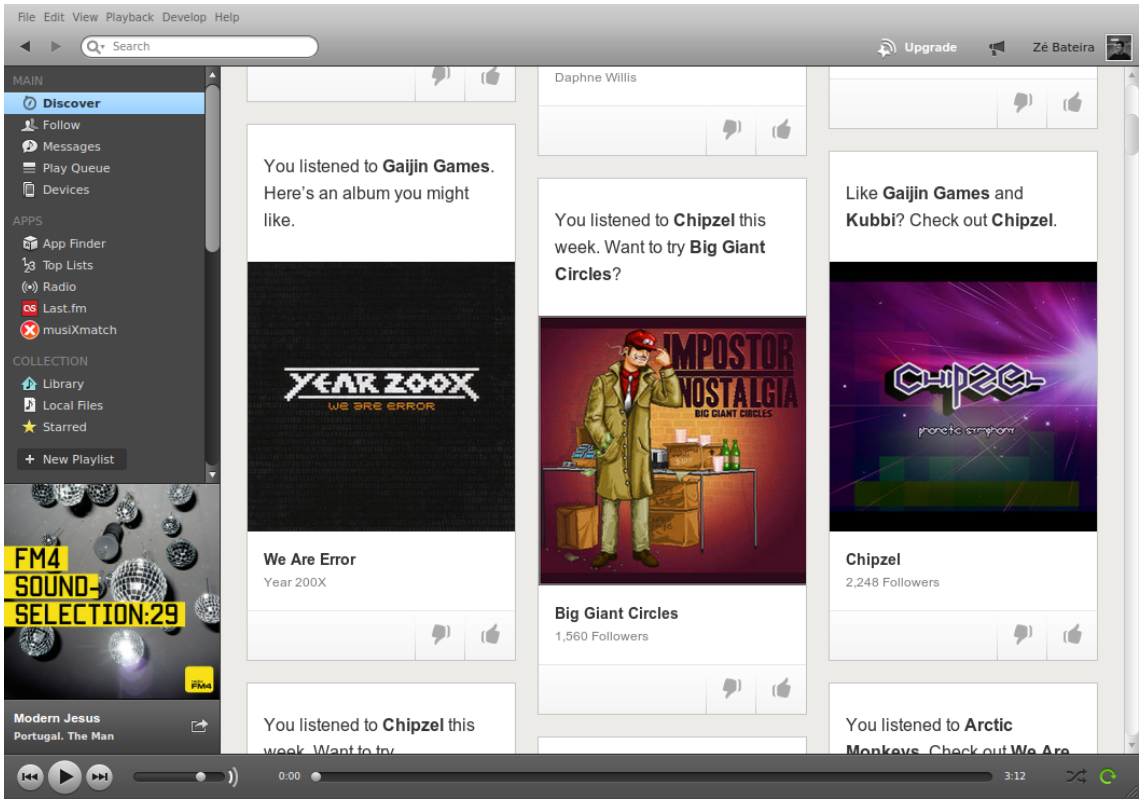


Figure 3.1: Spotify: desktop client's discovery mode interface.

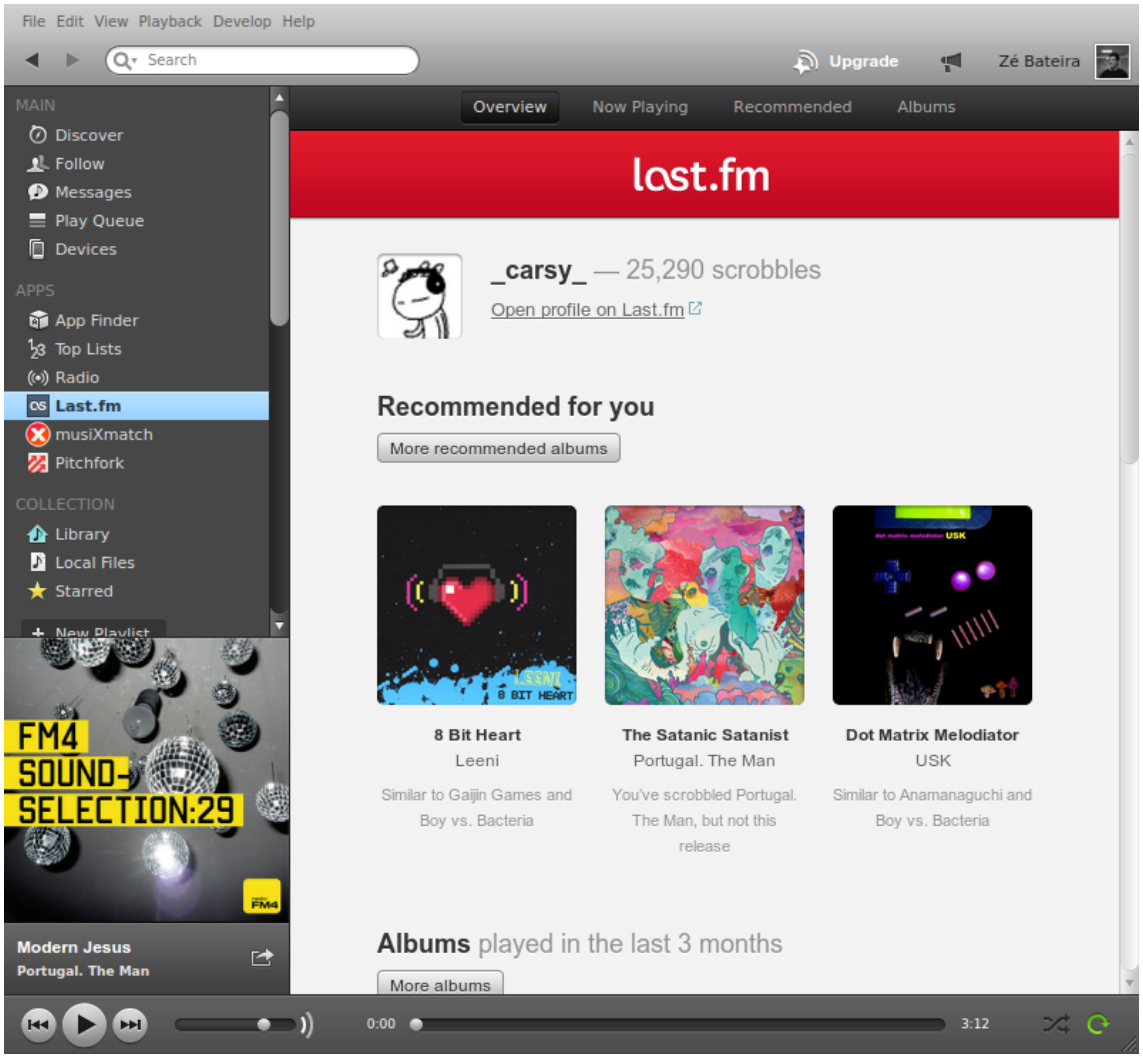


Figure 3.2: Spotify: Last.fm's Spotify Application opened.

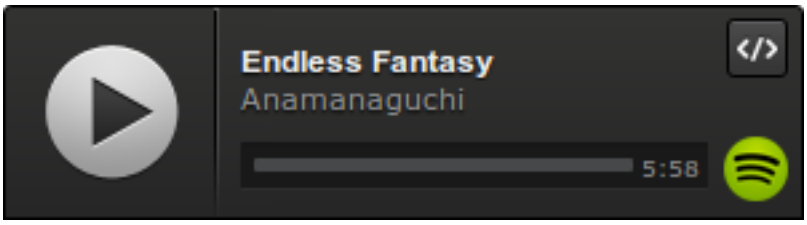


Figure 3.3: Spotify: *Play Button* pode ser embebido em *websites*.

The Applications' runtime environment is one of a browser-based. More specifically, powered by the Chromium Embedded Framework[3]. This means that the code to develop a Spotify Application follows the same principles as a web application: HTML, CSS and Javascript.

Spotify developed two Frameworks[4] to help developers create these applications: the API 1.x Framework[5] and the Views Framework[6].

The first one provides an interface to use object models, access metadata, control the player, among others. The second offers support for web components like buttons, lists, tabs, among others.

In order to develop the proposed modules 4 and 5, these are the most appropriate tools.

3.1.1.2 Spotify Widgets

As Widgets[7] 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.

[3] https://code.google.com/p/chromiumembedded
[4] https://developer.spotify.com/technologies/apps/reference
[5] https://developer.spotify.com/docs/apps/api/1.0/
[6] https://developer.spotify.com/docs/apps/views/1.0/
[7] https://developer.spotify.com/technologies/widgets

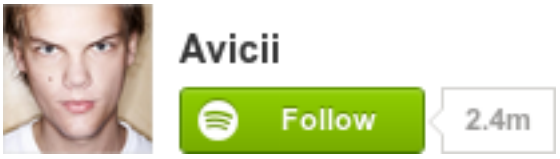


Figure 3.4: Spotify: *Follow Button* permite seguir um artista.

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*

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

3.1.1.3 Libspotify SDK

Libspotify SDK[8] 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*[9] disponibiliza publicamente informação de músicas, álbuns e artistas da Base de dados do Spotify.

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

Através de pedidos HTTP é possível obter informação da base de dados do Spotify. Existe dois tipos de pedidos que esta API disponibiliza: *search*[10] e *lookup*[11]. 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)

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,
```

[10] https://developer.spotify.com/technologies/web-api/search

[11] https://developer.spotify.com/technologies/web-api/lookup

```
        "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*.

As of the writing of this report, Spotify has a set of tools

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

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.

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.

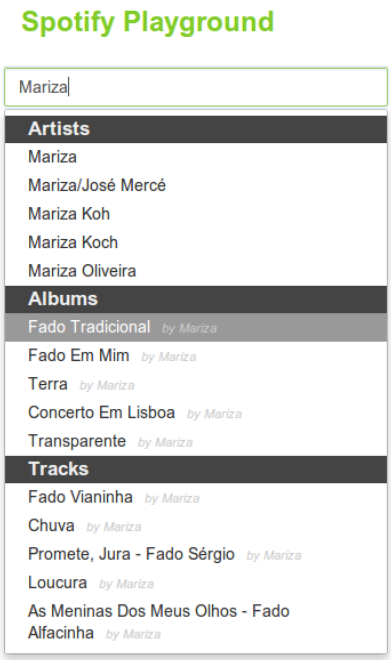
Assim, os módulos a serem desenvolvidos são 4 e 5.

3.2 Technologies used

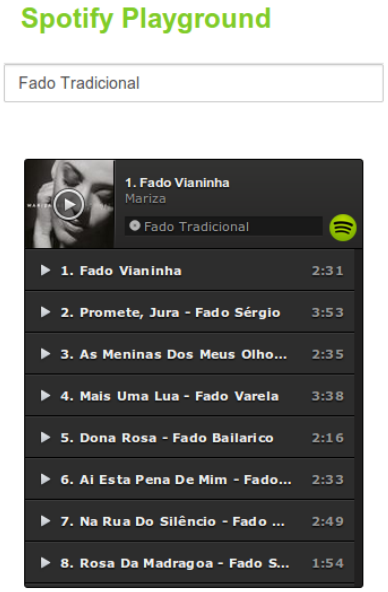
As seguintes tecnologias serão utilizadas nas fase de desenvolvimento, testes e otimização da Aplicação Spotify.

3.2.1 *Spotify Desktop Client*

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



(a) Resultado da pesquisa "Mariza"



(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`)

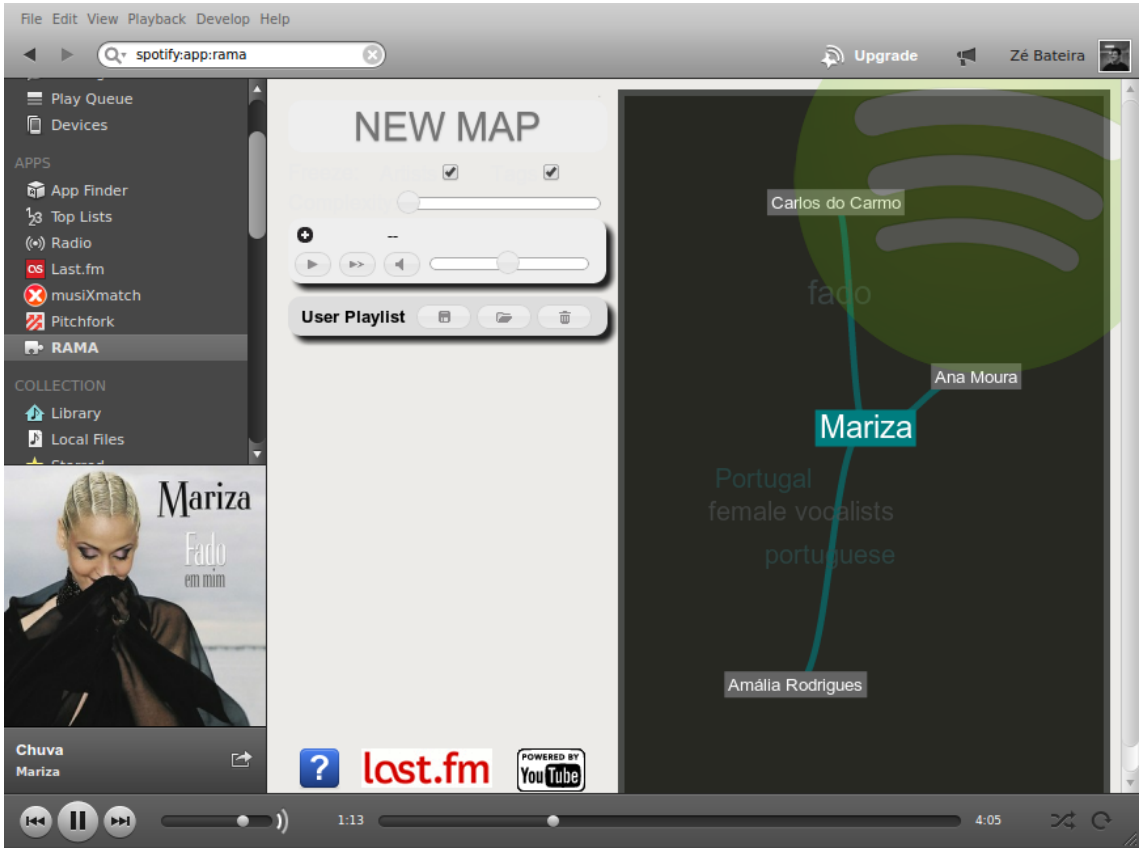


Figure 3.6: *Website* do RAMA embebido numa Aplicação Spotify

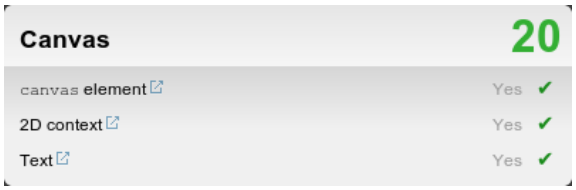


Figure 3.7: Resultado do teste do elemento *canvas*

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*[12].
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"
  },
  "SupportedDeviceClasses": ["Desktop"],
  "SupportedLanguages": [
    "en"
  ],
  "VendorIdentifier": "pt.inescporto"
}
```

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).

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

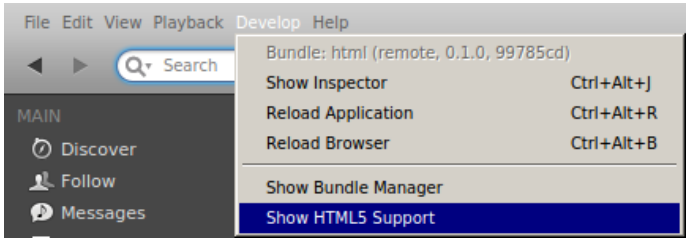


Figure 3.8: Menu *Develop*

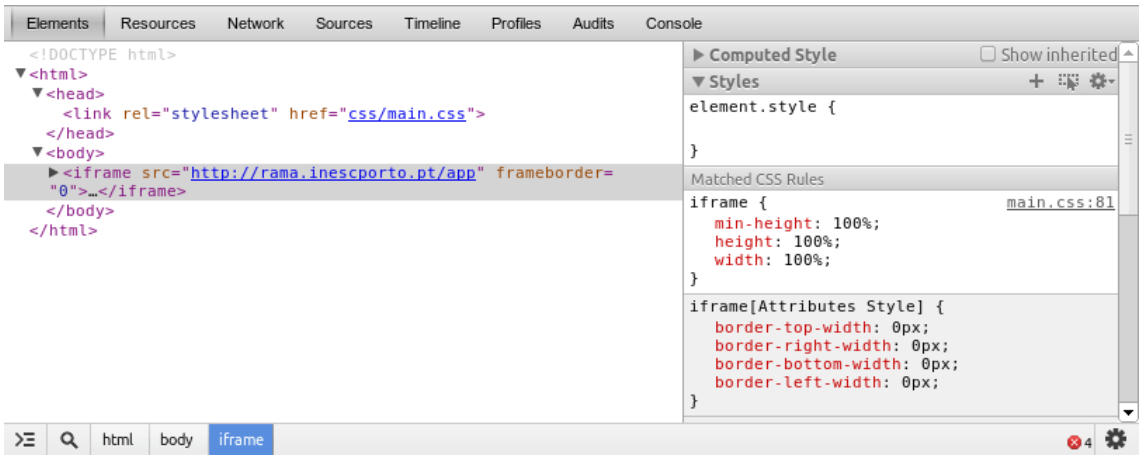


Figure 3.9: Webkit: Vista da tab *Inspector*. Outas ferramentas disponíveis (tabs): *Resources, Network, Sources, Timeline, Profiles, Audits* e *Console*.

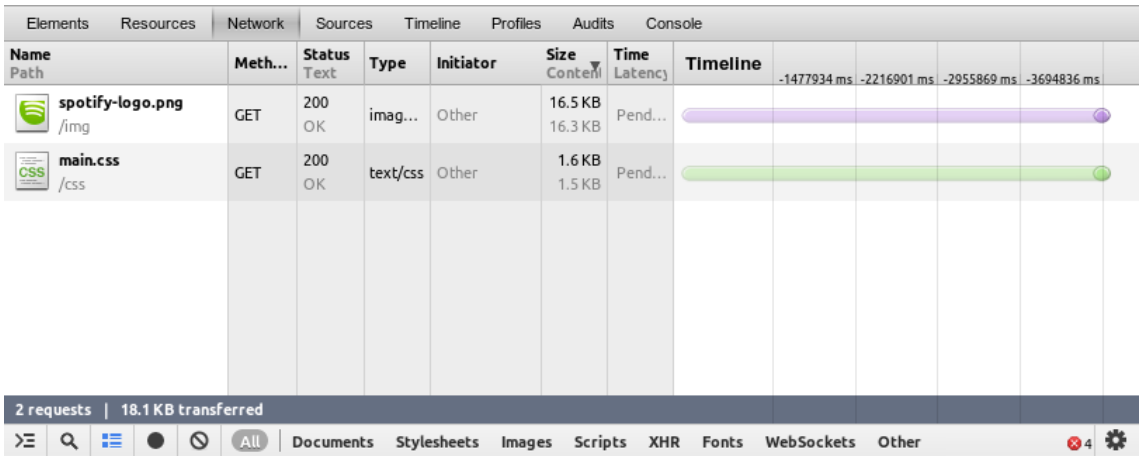


Figure 3.10: Webkit Network

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:

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

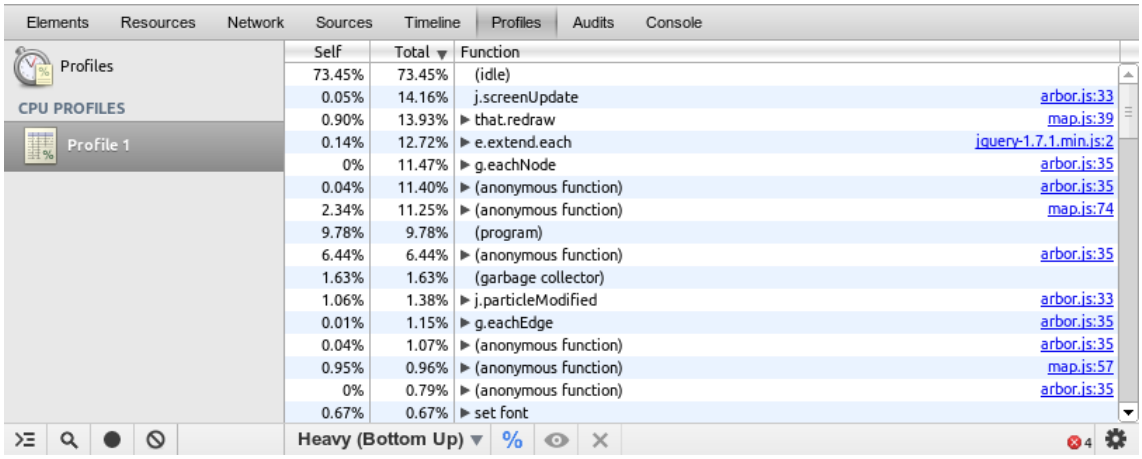


Figure 3.11: Webkit Profile: É possível ver que a renderização do grafo é o que ocupa mais tempo de processamento como esperado. No entanto, existe uma parte de *JQuery* que ocupa 12.72% do tempo de processamento, o que pode indicar um possível ponto de melhoria de performance.

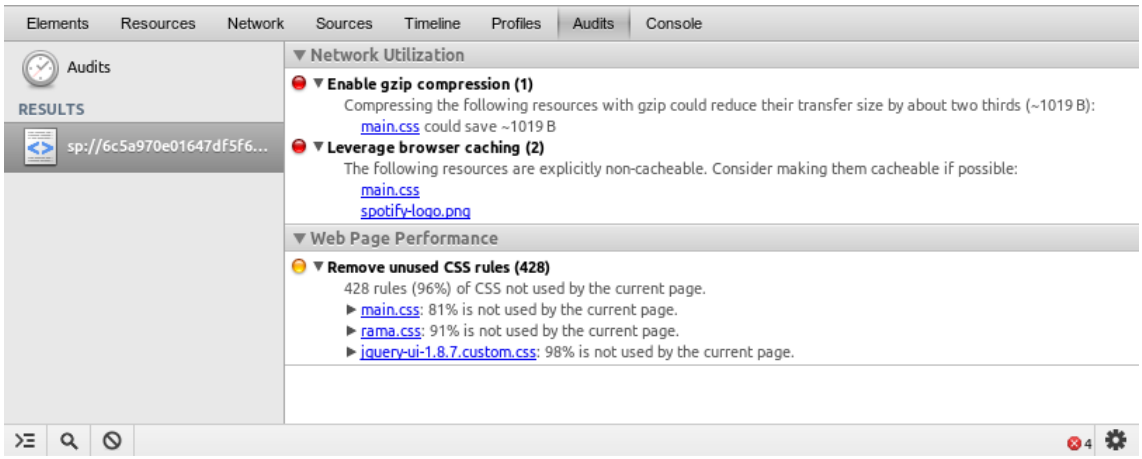


Figure 3.12: Webkit Audit: 96% do código *CSS* não está a ser usado, sendo por isso, um ponto de melhoria reduzir a quantidade de informação descarregada.

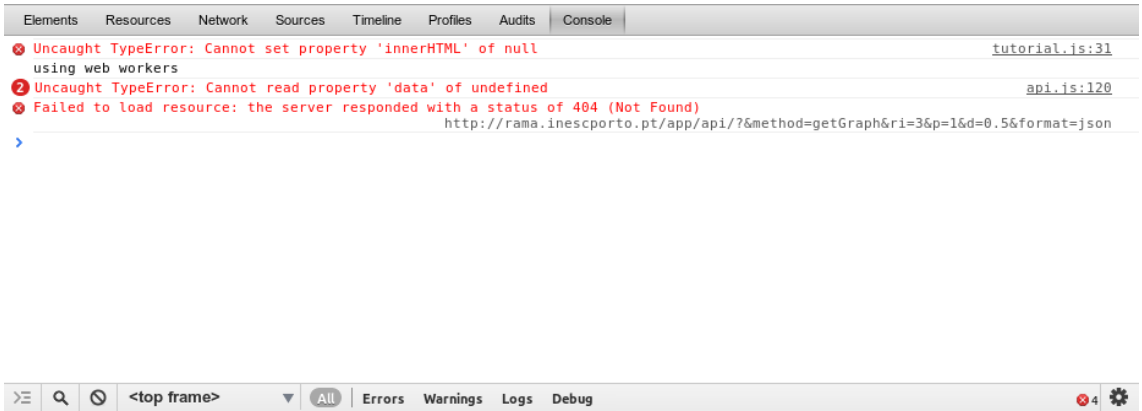


Figure 3.13: Webkit Console: Erros de *Javascript* aparecem destacados para chamar a atenção.

```
  "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 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 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.