

UNIVERSIDADE ESTADUAL PAULISTA

"JÚLIO DE MESQUITA FILHO"

Faculdade de Ciências - Campus Bauru

DEPARTAMENTO DE COMPUTAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

APLICATIVO WEB PARA ANÁLISE E COMPARAÇÃO DE AUDIO FINGERPRINT

BAURU

2015

EVANDRO BARBOSA CARREIRA

APLICATIVO WEB PARA ANÁLISE E COMPARAÇÃO DE AUDIO FINGERPRINT

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru.

Orientador: Prof. Dr. João Fernando Marar

Universidade Estadual Paulista “Júlio de Mesquita Filho”

Faculdade de Ciências

Ciência da Computação

BAURU

2015

Evandro Barbosa Carreira

Aplicativo web para análise e comparação de Audio Fingerprint/ Evandro Barbosa Carreira. – Bauru, 2015-

34 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. João Fernando Marar

Trabalho de Conclusão de Curso – Universidade Estadual Paulista “Júlio de Mesquita Filho”
Faculdade de Ciências
Ciência da Computação, 2015.

1. Audio Fingerprint 2. Open Source 3. Aplicação web I. Prof. Dr. João Fernando Marar. II. Universidade Estadual Paulista "Júlio de Mesquita Filho". III. Faculdade de Ciências. IV. Aplicativo web para análise e comparação de Audio Fingerprint

Evandro Barbosa Carreira

Aplicativo web para análise e comparação de Audio Fingerprint

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. João Fernando Marar
Orientador

Prof. Dr. Aparecido Nilceu Marana

Prof. Dr. Antonio Carlos Sementille

Bauru 2015

Espaço destinado à dedicatória do texto.

Agradecimentos

Espaço destinado aos agradecimentos.

Espaço destinado à epígrafe.

Resumo

Espaço destinado à escrita do resumo.

Palavras-chave: Palavras-chave de seu resumo.

Abstract

Abstract area.

Keywords: Abstract keywords.

Lista de ilustrações

Figura 1 – Diagrama de identificação de áudio.	17
Figura 2 – Ilustração da hélice de Shepard sobre percepção do <i>pitch</i> . O eixo vertical representa a altura do tom e a dimensão angular representa o <i>chroma</i>	20
Figura 3 – Representação da escala cromática iniciando em um Dó (C).	20
Figura 4 – Representação colorida de <i>chroma features</i>	21
Figura 5 – Formas de arranjo dos filtros visuais.	21
Figura 6 – Diferença bit a bit de duas versões da música, uma sem compressão e a outra comprimida em MP3.	22
Figura 7 – Diferença entre as versões originais e instrumentais de duas músicas.	22
Figura 8 – Ambiente de desenvolvimento da aplicação do modelo teórico.	25
Figura 9 – Teste de geração de hash simples.	29
Figura 10 – Organização do banco de dados do MusicBrainz.	30
Figura 11 – Geração de hash com algoritmo de <i>Chromaprint</i>	31
Figura 12 – Busca no banco de dados.	32

Lista de tabelas

Sumário

1	INTRODUÇÃO	13
1.1	Problema	13
1.2	Abordagem Colaborativa	14
2	OBJETIVOS	15
2.1	Objetivos Gerais	15
2.2	Objetivos Específicos	15
3	FUNDAMENTAÇÃO TEÓRICA	16
3.1	Audio Fingerprint	16
3.1.1	Pré-processamento	17
3.1.2	Enquadramento	17
3.1.3	Transformada	18
3.1.4	Extração de Características	18
3.1.5	Pós-processamento	18
3.1.6	Modelagem do Fingerprint	19
3.1.7	Busca e acesso ao BD	19
3.1.8	Teste de Acertos	19
3.2	Chromaprint	19
3.2.1	PCP - <i>Pitch Class Profile</i>	19
3.2.2	<i>Chroma</i> musical e a música ocidental	20
3.2.3	Comparação de fingerprint por filtro visual.	21
4	METODOLOGIA	23
4.1	Métodos e Etapas	23
4.2	Materiais Utilizados	23
4.2.1	Servidor de desenvolvimento	23
4.2.2	Github	23
4.2.3	V8 JavaScript Engine	23
4.2.4	Node.js	24
4.2.5	npm	24
4.2.6	FFmpeg	24
5	DESENVOLVIMENTO	25
5.1	Modelo Teórico e estrutura modular	25
5.1.1	Pré-processamento	26

5.1.2	Enquadramento	26
5.1.3	Transformada	27
5.1.4	Extração de Características	27
5.1.5	Pós-processamento	28
5.1.6	Modelagem do Fingerprint	28
5.2	Geração de hash com modelo de contagem de frequências	29
5.3	Base de dados e reconhecimento de músicas	30
5.4	Geração de hash <i>Chroma-based</i>	31
5.5	Busca, acesso e teste de acertos	31
6	CONCLUSÃO	33
	REFERÊNCIAS	34

1 Introdução

Em 1979, Tom Truscott e Jim Ellis da Universidade Duke criaram a Usenet (KAPLAN; HAENLEIN, 2010). Uma rede de comunicação onde as mensagens são agrupadas por assunto e exibidas em ordem cronológica de resposta, uma prévia dos fóruns virtuais que temos hoje. Como não existiam serviços de hospedagem de dados, a rede mantinha as mensagens armazenadas utilizando um conjunto de protocolos para propagar a informação de forma distribuída: Quando um usuário criava um artigo, ele ficava disponível somente na máquina deste usuário, cada máquina ligada à rede atualizava e mantinha as newsfeeds que possuía, fazendo com que o artigo criado fosse copiado diretamente em cada servidor até que estivesse presente em todas as máquinas ligadas à rede.

Esse formato de distribuição permite com que cada usuário mantenha em sua máquina somente os artigos referentes aos assuntos que lhe são de interesse, e fomenta a discussão em torno de assuntos específicos, eliminando a barreira física da curiosidade que existia até então. Segundo Lerner e Tirole (2000, tradução nossa) com a difusão da Usenet o processo de compartilhamento de software foi bastante acelerado e como o número de websites cresceu rapidamente (de 3 em 1979 para 400 em 1982), a habilidade dos programadores em compartilhar tecnologias foi consideravelmente melhorada.

Com o crescente aumento da velocidade de transferência de dados das redes, o compartilhamento passou de uma simples troca de informação para uma troca cultural, popularizando o acesso a versões virtuais de todo tipo de conteúdo audiovisual. Essa popularização alterou a forma como funcionava o mercado de consumo de músicas, vídeos e fotos e mudou a forma de como a produção cultural se desenvolveria futuramente, assim como todas as tecnologias que ainda estão se adaptando a esse novo modelo de compartilhamento.

1.1 Problema

O mercado da cultura musical das últimas décadas se manteve em um modelo onde a compra da música se limitava a um objeto físico que continha os dados do áudio, podendo ser reproduzido e até copiado, mas dependente do objeto para poder ouvir a música. Além de produto, a música é também comunicação intrapessoal, expressão emocional, parte da formação da cultura e do conceito de identidade do indivíduo. Com as facilidades da tecnologia além de mais fácil de consumir também ficou mais fácil de se produzir música. Uma música de 3 minutos comprimida em MP3¹ ocupa aproximadamente 3MB, tornando-a um bem praticamente

¹ Formato MPEG-1/2 Audio Layer 3, tipo de compressão de áudio com perdas quase imperceptíveis ao ouvido humano.

virtual por ocupar tão pouco espaço físico se comparado com os serviços de armazenamento disponíveis atualmente.

As tecnologias construídas para auxiliar o desenvolvimento desse novo mercado musical, devido aos modelos competitivos de propriedade intelectual, demoram mais para alcançar um número maior de pessoas. Uma dessas tecnologias ainda não disponíveis é o reconhecimento e agrupamento de músicas através do fingerprint do áudio. Existem aplicativos e softwares para download que fazem essa tarefa, mas não existe um framework que seja ao mesmo tempo: disponível para adaptação, tenha alto nível de modularização e de fácil compreensão.

1.2 Abordagem Colaborativa

Segundo Kaplan e Haenlein (2010, Tradução nossa), projetos colaborativos permitem a criação conjunta e simultânea de conteúdo por muitos usuários finais e são, nesse sentido, provavelmente a mais democrática manifestação de conteúdo gerado pelo usuário.

Ao desenvolver uma ferramenta que seja modular, é possível garantir uma maior coesão entre os componentes dessa ferramenta, pois cada módulo tem uma função definida e uma definição de formato de entrada e saída de dados.

Por se trabalhar com código aberto o conteúdo disponível é altamente modificável, pois o código faz parte da implementação e da transmissão do conhecimento. Essa possibilidade de modificação faz com que o código seja continuamente refatorado e revisto, e dentro de padrões de desenvolvimento tende a melhorar sua qualidade final.

Como diz Weber (2004, Tradução nossa) o código aberto não oblitera o lucro, o capitalismo, ou direitos intelectuais, e portanto não vai contra os modelos atuais de mercado, inclusive auxilia nos processos de todas as áreas sociais. O próprio formato do HTML² e essa ampla distribuição de código aberto através da internet tem como efeito uma grande divulgação de linguagens e ferramentas open source. Ferramentas que antes precisavam de um conhecimento mais amplo em computação, agora podem ser manipuladas com uma chamada do script direto no navegador de quem está testando um pedaço de código.

A utilização do formato modularizado garante também uma liberdade e uma intercomunicação entre trabalhos já desenvolvidos, como cálculos de transformadas e algoritmos de aprendizagem.

² HyperText Markup Language (Linguagem de Marcação de Hipertexto).

2 Objetivos

2.1 Objetivos Gerais

Desenvolver um aplicativo modular que contenha todas as funções de extração e comparação de audio fingerprints através de ferramentas web de código aberto.

2.2 Objetivos Específicos

- a) Estruturar o formato teórico para cálculo de diversos fingerprints;
- b) Modularização das fase de reconhecimento através do código;
- c) Geração de fingerprints musicais;
- d) Consultar base de dados para comparação e identificação;

3 Fundamentação Teórica

A utilização e implementação de bibliotecas externas, assim como a comunicação entre diferentes linguagens tem aproximado a interação que ocorre nas linguagens de programação com a interação de linguagens naturais.

A forma de transmissão dos dados caracteriza uma arquitetura de informação. Hollan, Hutchins e Kirsh (2000) dizem que:

“Processos cognitivos envolvem trajetórias de informações (de transmissão e transformação), de modo que os padrões destas trajetórias de informação, se estáveis, refletem uma arquitetura cognitiva subjacente. Uma vez que a organização social - mais a estrutura adicionada ao contexto da atividade - determina em grande parte o modo como a informação flui através de um grupo, a organização social em si pode ser vista como uma forma de arquitetura cognitiva.”

Dessa forma o desenvolvimento modular não se dá somente por convenção e facilidade de implementação, como também cumpre um papel de definição organizacional entre várias bibliotecas e estruturas de dados.

3.1 Audio Fingerprint

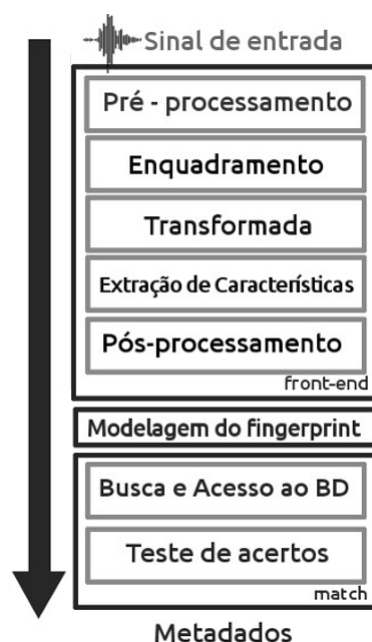
Um audio fingerprint é definido por Cano et al. (2005, tradução nossa) como uma assinatura única de uma música, contendo um sumário de suas características. Cano et al. (2005, tradução nossa) definem que um sistema ideal de fingerprint deve identificar um item, independentemente do nível de compressão, distorção ou interferência no canal de transmissão. E definem também que esse sistema pode ser separado em dois processos fundamentais: extração do fingerprint, e o algoritmo de comparação.

O processo completo de extração de dados pode ser dividido nos tópicos:

- a) Pré-processamento
- b) Enquadramento
- c) Transformada
- d) Extração de características
- e) Pós-processamento
- f) Modelagem do fingerprint
- g) Busca e acesso à base de dados

h) Teste de acerto

Figura 1: Diagrama de identificação de áudio.



Fonte: Elaborada pelo autor.

3.1.1 Pré-processamento

Para que o processamento e que a extração de características tome o menor tempo computacional possível, o sinal passará por um filtro que removerá dados que não interessam à extração de características, segundo Cano et al. (2005, tradução nossa): convertendo para o formato de 16 bits PCM¹, agrupando através da média os canais esquerdo e direito em uma taxa de amostragem variando de 5 a 44,1 kHz.

3.1.2 Enquadramento

O enquadramento adapta a amostragem para o que será utilizado. Cano et al. (2005) definem o enquadramento dizendo que:

“Um pressuposto fundamental na medição das características é que o sinal pode ser considerado como estacionário em um intervalo de alguns milissegundos. Portanto, o sinal é dividido em quadros de dimensão comparável à velocidade de variação dos eventos acústicos subjacentes. O número de quadros computados por segundo é chamado de taxa de quadros. Uma função de janela é aplicada a cada bloco para minimizar as discontinuidades entre o início e o fim. Uma justaposição

¹ Pulse-code modulation: método usado para representação digital de sinais analógicos.

deve ser aplicada para garantir robustez para o deslocamento (isto é, quando os dados de entrada não forem perfeitamente alinhados). E é necessário equilibrar os valores acima entre a taxa de variação no espectro e complexidade do sistema”

3.1.3 Transformada

A complexidade do sistema define como será tratado o enquadramento, para que o algoritmo de reconhecimento possa realizar uma melhor classificação dentro das características desejadas.

Os sinais sonoros que observamos são captados em função do tempo, mas ao aplicar transformadas que convertam esses sinais em função da frequência é possível observar particularidades que não podiam ser observadas anteriormente, assim como também é possível fazer um reconhecimento de padrões desses sinais em função da frequência. A transformada de Karhunen-Loève é uma transformada que consegue comprimir de maneira ótima os dados referentes ao sinal (Britanak, Yip e Rao, 2006), mas depende da função de entrada, tornando o cálculo computacional demasiadamente intensivo. Cano et al. (2005) também cita a decomposição em valores singulares como uma compressão ótima porém impraticável. Outras transformadas são citadas que podem ser utilizadas para extrair características:

- a) Transformada rápida de Fourier
- b) Transformada de Walsh-Hadamard
- c) *Modulated Complex Lapped Transform (MCLT)*
- d) Transformada Wavelet

3.1.4 Extração de Características

Os dados obtidos pela transformada serão novamente transformados de modo que aumente a sua invariância a distorções do som, como aumento do pitch ou mudança do tempo. Características que fazem parte do objetivo são levadas em conta nessa fase, como frequências relevantes (no caso de extração de voz ou reconhecimento de instrumentos), harmonicidade e ruído (no caso de classificação musical).

3.1.5 Pós-processamento

Para complementar os dados absolutos, as variações temporais podem ser adicionadas ao sinal processado, através das derivativas de primeira e segunda ordem.

Os métodos da primeira fase onde o sinal é pré-processado até aqui são agrupados e podem ser classificados como o front-end da obtenção do fingerprint.

3.1.6 Modelagem do Fingerprint

A modelagem do fingerprint usualmente recebe uma sequência de vetores de características calculadas quadro a quadro Cano et al. (2005, tradução nossa). As características que serão usadas para montar o modelo do fingerprint influenciam a forma como o algoritmo irá tratar futuramente a busca dessas características para comparação. Os processos mais simplificados como o de softwares como o *Music-brainz* traduzem as características para um hash de identificação, outros mais complexos armazenam o modelo em um formato de vetor ou matriz bidimensional.

3.1.7 Busca e acesso ao BD

A Busca depende diretamente do modelo definido e do problema a ser resolvido, pois nesse passo é preciso classificar o peso de cada característica extraída e a métrica utilizada para encontrar a distância entre cada fingerprint. O Banco de dados armazena os fingerprints já identificados e pode criar índices de busca para campos de maior relevância, para diminuir o custo computacional da comparação.

3.1.8 Teste de Acertos

Para identificação do fingerprint, ele passa por um teste de cálculo da probabilidade de acerto, onde é verificado se pertence ou não ao banco de dados atual, podendo ser classificado como pertencente ou não. Deve-se definir um limite de aceitação para classificar também um dado falso positivo (no caso de um banco de dados muito grande) ou um falso negativo (no caso de dados insuficientes).

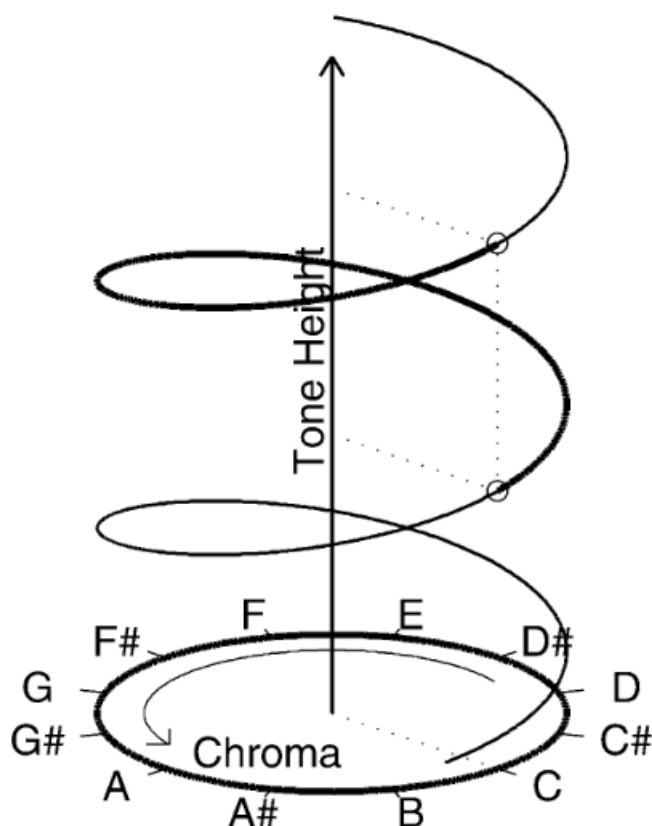
3.2 Chromaprint

Segundo Bartsch e Wakefield (2005, tradução nossa) a música popular é frequentemente baseada em uma estrutura simples que pode alternar entre versos e um refrão ou acorde repetido. Uma estratégia razoável para selecionar uma representação musical nestes casos mais simples envolve a localização e identificação destas seções que se repetem.

3.2.1 PCP - *Pitch Class Profile*

Shepard (1964, tradução nossa) diz que uma nota tocada nos remete a várias propriedades de percepção. Uma é o pitch, que é a qualidade do som determinada pela taxa de vibrações que produz; o grau agudo ou grave de um tom. Outra é a similaridade entre os intervalos de pitch, que também são chamadas de “cores” da nota, essa propriedade pode ser classificada como a escala de notas e denominada o *chroma* da nota, ou *pitch class profile*.

Figura 2: Ilustração da hélice de Shepard sobre percepção do *pitch*. O eixo vertical representa a altura do tom e a dimensão angular representa o *chroma*



Fonte: (BARTSCH; WAKEFIELD, 2005).

3.2.2 *Chroma* musical e a música ocidental

Na teoria musical ocidental o sistema de tons divide cada oitava em doze partes iguais, classificando a frequência de 440Hz como um lá (A), e progredindo a escala logaritmicamente a partir dessa frequência até repetir novamente a sensação cromática do lá, alcançada na frequência de 880Hz, esse intervalo entre as duas notas é denominado uma oitava, e dividido em 12 partes iguais, essa divisão é chamada de escala cromática.

Figura 3: Representação da escala cromática iniciando em um Dó (C).



Fonte: Elaborada pelo autor.

Devido a repetição de frequência e similaridade de notas, podemos representar as notas de uma música em um vetor de 12 posições, um para cada classe de *pitch*. Essa representação nos permite extrair características musicais através de uma DFT²:

Figura 4: Representação colorida de *chroma features*.



Fonte: (LALINSKY, 2011).

3.2.3 Comparação de fingerprint por filtro visual.

(JANG et al., 2009) definem um algoritmo de comparação de dois *audio fingerprints* com base em um filtro visual de soma de características. Existem dezesseis tipos de filtros aplicados a uma janela de tamanho de 16x12 pixels, cada filtro é aplicado para capturar diferentes intensidades musicais.

Figura 5: Formas de arranjo dos filtros visuais.



Fonte: (LALINSKY, 2011).

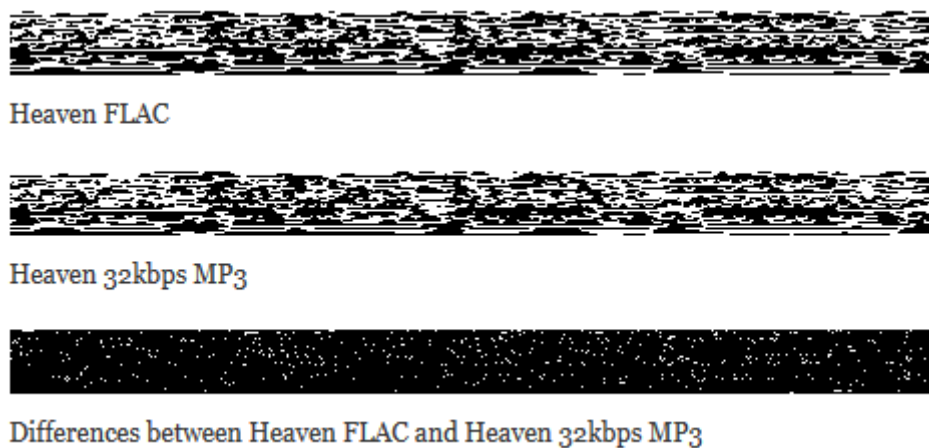
Segundo Lalinsky (2011, tradução nossa) para cada filtro é calculada a diferença da soma das áreas brancas com a soma das áreas pretas. Cada filtro tem três coeficientes associados a ele, que guiam como classificar o resultado de 0 a 3. Esses filtros e coeficientes foram selecionados por um algoritmo de aprendizado de máquina, treinado durante o desenvolvimento da biblioteca.

O resultado dos 16 filtros produz um inteiro que pode ser codificado em 2 bits, combinando os resultados temos um inteiro de 32 bits, se aplicarmos estes filtros para cada janela dos *chroma features* obtemos um *audio fingerprint* completo.

Com base nesse fingerprint a busca por proximidade em um banco de dados pode ser otimizada para encontrar uma música que esteja mais próxima da música procurada, mesmo que em versões diferentes.

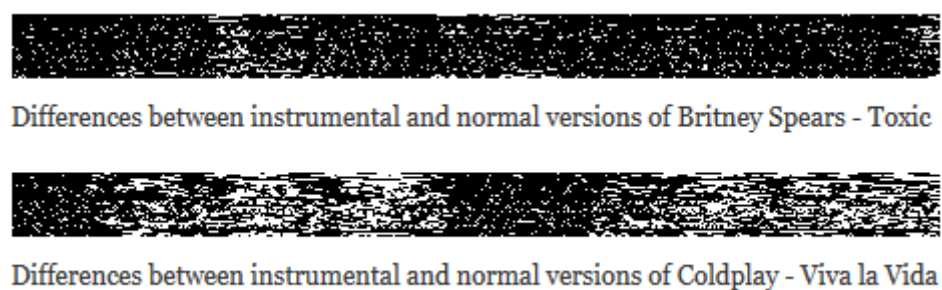
² Discrete Fourier transform - Transformação discreta de Fourier

Figura 6: Diferença bit a bit de duas versões da música, uma sem compressão e a outra comprimida em MP3.



Fonte: (LALINSKY, 2011).

Figura 7: Diferença entre as versões originais e instrumentais de duas músicas.



Fonte: (LALINSKY, 2011).

4 Metodologia

4.1 Métodos e Etapas

Para o desenvolvimento do projeto foi realizado o levantamento bibliográfico de tecnologias existentes e modelos organizacionais referentes à extração de *Audio Fingerprint*, assim como a comparação específica de modelos de ondas sonoras de músicas e melodias.

Com a base teórica definida a etapa seguinte foi aplicar uma estrutura modular que correspondesse ao modelo teórico, para que todas as etapas da extração e reconhecimento do *audio fingerprint* possam ser modificadas sem que alterem o funcionamento geral do processo. Essa estrutura adaptável foi aplicada com base no modelo teórico genérico de reconhecimento de áudio.

A terceira e última etapa foi o desenvolvimento de uma aplicação de reconhecimento específico de músicas, utilizando um cálculo de *audio fingerprint* baseado em representações que levam em conta a repetição de refrões através da análise baseada em altura tonal e *chroma* musical.

4.2 Materiais Utilizados

4.2.1 Servidor de desenvolvimento

Para desenvolvimento do projeto foi utilizada uma máquina externa com 512MB de RAM, Disco SSD com 20GB e Ubuntu 14.04 x64.

4.2.2 Github

O Github é ao mesmo tempo um servidor de armazenamento de código e uma rede social onde pode-se submeter modificações, fazer cópias e acompanhar modificações de códigos de outras pessoas. A rede foi essencial para o desenvolvimento deste projeto por armazenar vários sub-módulos e disponibilizar código-fonte para consulta.

4.2.3 V8 JavaScript Engine

V8 é um projeto open-source da Google para interpretação de Javascript, escrito em C++ e utilizado no navegador *Google Chrome*. Ele implementa o ECMAScript e pode ser inserido em aplicações C++ nativas.

4.2.4 Node.js

O Node.js é um interpretador *JavaScript* multiplataforma que utiliza o *V8 JavaScript Engine*, foi criado em 2009 por Ryan Dahl e atualmente é utilizado para aplicações variadas. Seu funcionamento *single threaded* simplifica a construção da aplicação, foi escolhido para o desenvolvimento deste projeto por facilitar o desenvolvimento modular e comunicação entre os módulos.

4.2.5 npm

O npm é uma sigla para *node package manager* mas é um gerenciador de pacotes geral que implementa não somente os pacotes de node como também de projetos variados em javascript e outras linguagens. O npm permite a instalação e gerenciamento de pacotes e bibliotecas de maneira facilitada, coordenando as versões utilizadas de acordo com a necessidade do projeto.

4.2.6 FFmpeg

O FFmpeg é um programa externo multiplataforma utilizado através do node. Ele grava, converte e cria *stream* de vários formatos de áudio e vídeo.

5 Desenvolvimento

5.1 Modelo Teórico e estrutura modular

Seguindo a abordagem descrita por Cano et al. (2005) foi desenvolvido um módulo para cada fase dos processos de extração do *audio fingerprint*:

- a) Pré-processamento
- b) Enquadramento
- c) Transformada
- d) Extração de características
- e) Pós-processamento
- f) Modelagem do fingerprint

Cada etapa é um módulo independente que passa a informação tratada através de stream de audio ou arquivo para a próxima etapa até a obtenção do *audio fingerprint* final. Uma etapa anterior foi adicionada para extrair o audio de uma fonte web e enviar para o sistema de geração de *audio fingerprint*.

Figura 8: Ambiente de desenvolvimento da aplicação do modelo teórico.

```
server.js+ $ > package.json+ {}  
.. (up a dir)  
/home/evandro/dev/audiofingerprint/  
node_modules/  
  fluent-ffmpeg/  
    doc/  
    lib-cov/  
    lib/  
    node_modules/  
    tools/  
      build.bat  
      Gruntfile.js  
      index.js  
      LICENSE  
      Makefile  
      package.json  
      README.md  
  fpcalc/  
  pcm-utils/  
  youtube-audio-stream/  
    musica  
    musicacopia  
    package.json  
    pcmmono  
    pcmraws16le  
    server.html  
NERD  
1 {  
2   "name": "audiofingerprint",  
3   "description": "audiofingerprint",  
4   "version": "0.0.1",  
5   "main": "index.js",  
6   "keywords": [  
7     "audio",  
8     "fingerprint"  
9   ],  
10  "dependencies": {  
11    "fluent-ffmpeg": "*",  
12    "fpcalc": "^1.1.2",  
13    "digitalsignals": "*",  
14    "pcm-utils": "evandrodo/node-pcm-utils",  
15    "youtube-audio-stream": "*" 16  }  
package.json[+] 76% 13: 27  
server.js+  
2 var ffmpeg = require('fluent-ffmpeg');  
3 var fs = require('fs');  
4 var streamify = require('youtube-audio-stream');  
5 var fpcalc = require('fpcalc');  
6 var through = require('through2');  
7 var DFT = require('digitalsignals');  
8  
9 // Crio um server escutando na porta 3420  
10 http.createServer(requestListener).listen(3420);  
11  
12 /**  
13  * Trata request inicial e escuta o audio que  
14  * será enviado para a primeira etapa.  
15  */  
16 +- 13 lines: function requestListener(req, res) {-----  
29  
30 /**  
31  * Pré processa o Audio, remove dados que não interessam à extração  
32  * de características convertendo para o formato de 16 bits PCM,  
33  * agrupando através da média os canais esquerdo e direito em  
34  * uma taxa de amostragem variando de 5 a 44,1 kHz.  
35  */  
36  * @param {file} file - Stream de audio recebida da fase de escuta  
37  */  
38 +- 26 lines: function preProcessing(file) {-----  
64  
65 /**  
66  * Aplica uma função de janela no sinal cru pré processado, para minimizar  
67  * os cálculos. Quanto maior o frame rate, mais exato fica o sistema e  
68  * maior é o custo computacional.  
69  */  
70  * @param {PCM 16bit file} file - Arquivo PCM em 16bit e um Único canal  
71  */  
72 +- 13 lines: function framingAndOverlap(file) {-----  
85  
86 /**  
87  * Converte o áudio filtrado do domínio do tempo para o domínio da frequência,  
88  * A próxima etapa (Extração de características) é bastante dependente da  
89  * forma como os dados serão tratados aqui.  
90  */  
91  * @param {PCM 16bit file} file - Arquivo PCM em 16bit e um Único canal  
92  */
```

Fonte: Elaborada pelo autor.

5.1.1 Pré-processamento

Pré processa o Audio, converte para o formato de 16 bits PCM, e agrupa através da média os canais esquerdo e direito em uma taxa de amostragem variando de 5 a 44,1 kHz.

```
function preProcessing(file) {
    // Cria um stream de leitura com base no arquivo que está
    // recebendo a musica em PCM
    var preProcStream = fs.createReadStream(file),
        pcmMono = fs.createWriteStream('pcmmono'),
        musica = '';

    preProcStream.on('data', function(chunk) {
        console.log(chunk.length);
        musica += chunk;
    });

    preProcStream.on('end', function(chunk) {
        console.log('Iniciando conversão para PCM...');
        ffmpeg(__dirname + '/musicacopia')
            .format('pulse')
            .audioChannels(1)
            .output(__dirname + '/pcmmono')
            .on('end', function() {
                console.log('Música convertida para PCM 16bits Mono');
                framingAndOverlap(__dirname + '/pcmmono');
            })
            .run();
    });
}
```

5.1.2 Enquadramento

Aplica uma função de janela no sinal cru pré processado, para minimizar os cálculos. Quanto maior o frame rate, mais exato fica o sistema e maior é o custo computacional.

```
function framingAndOverlap(file) {
    var preProcStream = fs.createReadStream(file),
        frameSize = 4096,
        overlap = 75, //porcentagem
```

```

    fao = new framingAndOverlap();

    fao.size(frameSize).overlap(overlap).run(preProcStream);
    transformada(file);
}

```

5.1.3 Transformada

Converte o Audio filtrado do domínio do tempo para o domínio da frequência, a próxima etapa (Extração de características) é bastante dependente da forma como os dados serão tratados aqui.

```

function transformada(file) {
    var procSinal = fs.createReadStream(file),
        dftStream = fs.createWriteStream(__dirname + '/dft'),
        dft = new DFT(1024, 44100);
    dft.forward(procSinal);
    var spectrum = dft.spectrum;
    spectrum.pipe(dftStream);
    extracao(dtStream);
}

```

5.1.4 Extração de Características

A extração de características deve ser ajustada para a finalidade desejada do *audio fingerprint*. No caso de reconhecimento de voz ou instrumentos, os dados mais relevantes são os referentes às frequências com maior índice de repetição (harmônicas presentes). No caso de reconhecimento de músicas os dados relevantes são a repetições de refrões, melodia e *chroma* musical.

```

function extracao(stream) {
    var sinalfreq = fs.createReadStream(stream);

    // Análise simplificada do sinal sonoro: contando as frequencias
    // que mais se repetem (harmonicas de instrumento)
    sinalfreq.on('end', function(chunk) {
        console.log('Contando frequências do sinal e agrupando');
        var coordFN = sinalfreq;
        // Envia a contagem para o pós processamento como um vetor
        // de distância em N dimensões, sendo N o número de

```

```

        // frequências disponíveis
        posprocess(coordFN);
    });
}

```

5.1.5 Pós-processamento

Para gerar um fingerprint o dado com as características relevantes deve ser normalizado e pode ser simplificado através de derivadas de primeira e segunda ordem.

```

function posprocess(coords) {
    var maxFreq = -11000,
        minFreq = 11000;
    for(int i=0;i<coords.length;i++) {
        if(coords[i]<minFreq) {
            minFreq = coords[i];
        }
        if(coords[i]>maxFreq) {
            maxFreq = coords[i];
        }
    }
    var amplitude = maxFreq - minFreq,
        intervalo = amplitude/36; //26 letras e 10 numeros
    // Normalizando o sinal para 36 representações
    for(int i=0;i<coords.length;i++) {
        coords[i] = Math.round(coords[i]/intervalo);
    }
    modeling(coords);
}

```

5.1.6 Modelagem do Fingerprint

A modelagem do fingerprint usualmente recebe uma sequência de vetores de características. Através desse vetor é gerado um hash de identificação.

```

function modeling(coords) {
    var simbolos = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l",
        "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",
        "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"];
    hash = "";
}

```

```

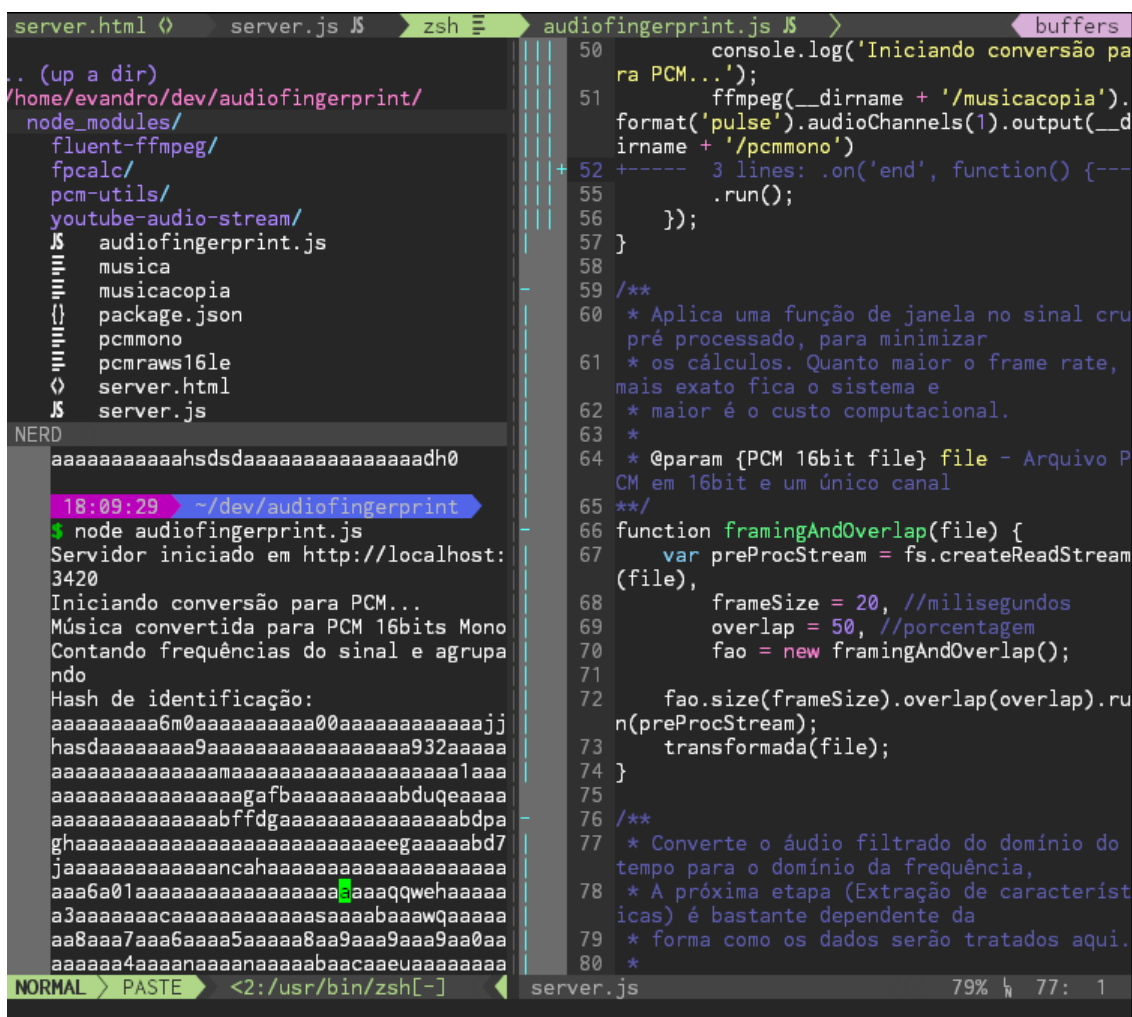
for(int i=0;i<coords.length;i++) {
    hash = hash + simbolos[coords[i]];
}
console.log('Hash de identificação:');
console.log(hash);
}

```

5.2 Geração de hash com modelo de contagem de frequências

Com base no algoritmo modular desenvolvido foi gerado um hash para teste de eficiência de processamento, esses módulos podem ser modificados para atender vários modelos de *audio fingerprint*. Como exemplo as transformadas DFT podem ser substituídas por transformadas Wavelet e o vetor pode ser resumido à derivativa de primeira ordem, dependendo da finalidade da aplicação.

Figura 9: Teste de geração de hash simples.



```

server.html ◀ server.js JS zsh ▶ audiofingerprint.js JS ▶ buffers
. (up a dir)
/home/evandro/dev/audiofingerprint/
node_modules/
  fluent-ffmpeg/
  fpcalc/
  pcm-utils/
  youtube-audio-stream/
  JS audiofingerprint.js
  musica
  musicacopia
  {} package.json
  pcmmono
  pcmraws16le
  ◀ server.html
  JS server.js
NERD
aaaaaaaaaahsdsdaaaaaaaaaaaaaadho
18:09:29 ~ /dev/audiofingerprint
$ node audiofingerprint.js
Servidor iniciado em http://localhost:
3420
Iniciando conversão para PCM...
Música convertida para PCM 16bits Mono
Contando frequências do sinal e agrupa
ndo
Hash de identificação:
aaaaaaaaa6m0aaaaaaaaaaaa00aaaaaaaaaaaaajj
hasdaaaaaaaaa9aaaaaaaaaaaaaaaaaaaa932aaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa1aaa
aaaaaaaaaaaaaaaaaaaaafbbaaaaaaaaaabduqaaaaa
aaaaaaaaaaaaaaaaabffdgaaaaaaaaaaaaaabdpaa
ghaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaegaaaaabd7
jaaaaaaaaaaaaancahaaaaaaaaaaaaaaaaaaaaaa
aaa6a01aaaaaaaaaaaaaaaaaaaaaqaqwehaaaaaa
a3aaaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaawqaaaaa
aa8aaa7aaa6aaaa5aaaaa8aaa9aaa9aaa9aaa0aa
aaaaaaa4aaaaaanaaaaaabaaacaaeuaaaaaaaaa
NORMAL > PASTE <2:/usr/bin/zsh[-] server.js 79% 77: 1

```

```

50 console.log('Iniciando conversão pa
ra PCM...');
51 ffmpeg(__dirname + '/musicacopia').
format('pulse').audioChannels(1).output(__d
irname + '/pcmmono')
52 +----- 3 lines: .on('end', function() {---
55 .run();
56 });
57 }
58
59 /**
60 * Aplica uma função de janela no sinal cru
pré processado, para minimizar
61 * os cálculos. Quanto maior o frame rate,
mais exato fica o sistema e
62 * maior é o custo computacional.
63 *
64 * @param {PCM 16bit file} file - Arquivo P
CM em 16bit e um único canal
65 */
66 function framingAndOverlap(file) {
67     var preProcStream = fs.createReadStream
(file),
68     frameSize = 20, //milisegundos
69     overlap = 50, //porcentagem
70     fao = new framingAndOverlap();
71
72     fao.size(frameSize).overlap(overlap).ru
n(preProcStream);
73     transformada(file);
74 }
75
76 /**
77 * Converte o áudio filtrado do domínio do
tempo para o domínio da frequência,
78 * A próxima etapa (Extração de característ
icas) é bastante dependente da
79 * forma como os dados serão tratados aqui.
80 *

```

Fonte: Elaborada pelo autor.

5.3 Base de dados e reconhecimento de músicas

5.4 Geração de hash *Chroma-based*

Para a geração de hash para consulta de músicas foi utilizado o algoritmo de *Chromaprint*, descrito na seção 3.2. O algoritmo gera um hash representado em caracteres alfanuméricos que pode ser utilizado para consulta nos bancos de dados disponíveis.

Figura 11: Geração de hash com algoritmo de *Chromaprint*.

```

10
11 /**
12  * Trata request inicial e escuta o audio que
13  * será enviado para a primeira etapa.
14  */
15 function requestListener(req, res) {
16   if (req.url === '/') {
17     return fs.createReadStream(__dirname + '/server.html')
18       .pipe(res);
19   }
20   if (/youtube/.test(req.url)) {
21     var opt = {
22       file: __dirname + '/musica'
23     };
24
25     streamify(req.url.slice(1)).pipe(res);
26     streamify(req.url.slice(1), opt)
27       .on('pipe', function() {
28         console.log('Calculando Chromaprint...');
29       })
30       .on('finish', function() {
31         console.log('acabou o stream');
32         fpcalc(__dirname + '/musica', function(err, result) {
33           if (err) throw err;
34           console.log('Duração da música: ');
35           console.log(result.duration+' segundos');
36           console.log('Audio Fingerprint: ');
37           console.log(result.fingerprint);
38         });
39       });
40   }
41 }
42
43 function geraHash(file) {
44   var writableStream = fs.createWriteStream(__dirname + '/music
45   a');
46   writableStream.on('finish', function(chunk) {
47     console.log('acabou o stream');
48   });
49   /*
50     fpcalc(__dirname + '/musicahash', function(err, result) {
51       if (err) throw err;
52       console.log(result.file, result.duration, result.fi@@
53
54 ver.js javascript JS 20% 28: 50

```

```

node server.js
e relaxa http://localhost:3420
Calculando Chromaprint...
acabou o stream
Duração da música:
516 segundos
Audio Fingerprint:
AQADtIIMRSofjSwR5hlyHxUVVtBRKzxyCt1z-OHgFx_-4zre44Qr_EdF9
LhzoTq-I0FOfMnxo94MPzhiU4c2FCPy4kFIwxZRXIul8B0s5C80HeHxYf
dqH8ePHleRH_rRQx9--Lhw1Dv-4Id-HMWPAP8eB4_jx4fjx44J6yPiz4Cc
xOVfBID0mhovxFT-RFrg4Qg_foVYeuAkzBcfJ7SgfpNL540-hB-P8pDx6
9Be8Bz-G7_BxQMdx_LBxHT-079BhCWeKHP3h44gvpiifJRJe4KIOGycur
-j2480P52h-_MI3WDj-4Thu4XiPH-eHh0X9IzqYH-kUEW00Jk-k079BP9F
GMLWSOT80J_fwH80z5MkX6DSR48vhP8HxJMcbwQ-i8-Ax_kW77AF8HNd
x_OhxAFen2kezojFMctfRHF9WnBV--Ap-Bc80PbZx9Gi-4zu-Qbu0s8U3
_LCTHX1zXA_es0iP82g24yKe5Tiu4h_OMBr-MjjaHH7wdfh2dH_w40f7C
s1L9D-af8Jxn0iFh8cPHPiB480PDzh-HD_-ES_uc3n6EK4_1DzmLDMYxw
hTC6N16Jzw7ETnVFHQyDnuTKjYwh-P5sugOPux9DiTHNBR5uhnHPPBY9f
RDWnQFI60o6kScbiJn3hx4AAgHD8AADnw4_iBP8dxhTxy3fihvTSOK0d0
Fz3Gw0e9JEGPxxnA5sd3Er0YwaHTo-uhHteLrwv8NMSXD1fhHBcvHH-wF
5f1ZMEj_A16s2jkHG-OKUuS5Ao80jncK9C-wN0CMotkVO1D_GLw5cJ9wX
uNT4mHzwGnB496qKp01Doaf89j406uQlfi41SG6U_gzBGFfoQ80fj0BG-
OhpWkP0iPRgoVqi_w50ijC64jnfXwGJodI6eLP9zQSC0x00eYXzh96Gjy
HqGVI-81PEd_FcqyHH0zRKSW4xc65YGtwZPTIPwL7UKYTiXYjRze7EE1p
JvwGzqyyInwXXj04sef4o0qNJO08FCnsA7xJDf6xGjIHeMT0KHJwq60k
Grq_iDiWzx021E-UyYFcic0PR90lQynEE_WCC0AqZ4NaPP8XlxHiWeTi
dDE0_hIFmJUGu3bjUhBG-zMMoPsGdbppQSPmDizn0fMhz9D18J2BEFQ8X
Dbup4EFtHW5E42eE9DxUZ5vQ6DH15cat4CEtwg1nb8mVVA-hkfVqah4S
ImaG90XAsPzcol-Egj2egYKRWaLMqF8qqQHfQYUyoZHI7WHZayDxp9xB
6eqCqp2iid_i0h7mwG01k90yRJ4fQZ8j54VXBK0d1NKUSHQ-NO_rQN3g
zzPmD6g0a-9gpJkaf_HadHeeM5vgh-6gzxREsxQ36D_Hj4heankX0QteD
lEuJ_hj9Y3eIR8h3aGmke0jEBz-Kw32DB-2Ln0ZRL46CHnfQNBwPfiwb
YpQua3xD2_xH-cBXLrwIxc0T1mQxkVWUR3-Cg1VonwqHFcQ5sxQHuoYSX
hznMnIoZmco5eQF02oC-Gi6Au0nEU4Xc0kH-ZoHVe0C_4Kjx-u8EgJmMq
LFpI3g724UeeQxyHcd3nL7wM7wHs9x0cIv-A-0-zBz5A9-NDGRUwme
4y9-S8jzBrt4-0C-TcP08Sp0Dc-HHg0jhngyGL-fjhcZ9-iPH8cZ0gbzD
E10HCenIGdEIVuVdP8ZgkHrkz4j1-fzBhMlk4RRj42kUra-eLcfnoW
GMkKxxZ8SPH3UCw8v4kR-XIei6zhV8Dxx7YRSNoaZRUAxDt_xIFyUIw_
XRNjGsDjBPS3-BvsiQ8d5pMtyXBSufLDTJg2070G045mhPQ54Fz8aSGN
H1ouZrReC9-QSwaG5reoskUNmAs5gzC52h-jNnxD-9S4duF-kEZsERP
_licwy4VPD2uDLWW1Gh0BXqkoL2Qksf7FX5PfiFjCxMTyogfQs-RR0Z3fm
Wi9riUHZ-J5gjPQB-MPT_06fjy4NBtpJrCgMeOoy-eJ9jeIKSaHqmbaE
fool8fPlxPEH25WicQRP1JWCZHJr2w3vxPZDuePhx6sijKm8-_MFzHvyD

```

Fonte: Elaborada pelo autor.

5.5 Busca, acesso e teste de acertos

Com o hash representando o *audio fingerprint* o último passo necessário para o reconhecimento é a consulta no banco de dados. Para a busca, acesso e teste de acertos foi utilizada a base disponibilizada (MUSICBRAINZ, 2015) e o *web service* da AcoustID, com acesso e comparação através do hash enviado:

Figura 12: Busca no banco de dados.

```

16:22:08 X ~/dev/audiofingerprint
$ node server.js
Servidor executando em http://localhost:3420
Calculando Chromaprint...
acabou o stream
Duração da música:
516 segundos
Audio Fingerprint:
AQADtILMRSoFjSwR5hlyHxUVVtBRKzxyCt1z-OHgFx_-4zre44Qr_Edf9LhzoTq-I0FOFmXo94MPzhiU4c2fCPy4kfIwxZRXIuL8B0s5C80HeHxHydQh8ePH1eRH_rRQx9-
-Lhw1Dv-4Id-HMwPA8eB4_jx4fjx44J6yPiz4CcxOVfBID0mhovxFT-Rfrq4Qg_foVYeuAkzBcfJ7SgfPNLS40-hB-PBpDx69Be8Bz-G7_BxQMdx_LBxHT-079BhCWeKHP3h
44gvpiifJRJe4KIOGycur--j2480P52h-_MI3WDj-4Thu4XiPH-eHh0X9IzqYH-kUEW00Jk-079B9PFGCMLWS0T80J_fwH80z5MkX6DSR48vhP8HxJmcbwQ-i8-Ax_kW77AF8
HNdx_OhxAFen2kezojfmCtFRHF9WnBV--Ap-Bc80PbZx9G1-4zu-Qbu0s8U3_LCTHX1zXA_es0iP82g24yKe5TiU4h_OMBr-MjjaHH7wdfh2dH_w40f7Cs1L9D-af8Jxn0iF
H8cPHPiB480PDzh-HD_-ES_uC3n6EK4_1DzmLDMYxwhTC6N16Jzw7ETnVFHQYDnuTKjYwh-P5sug0Pux9DITHN8R5uhnHPPBY9FRDwnQfIG0o6kScbiJn3hx4AAgH8DAADnw
4_ibP8dxhTxy3fihvTSOK0d0Fz3Gw0e9JEGPxnNA5sd3Er0YwahTo-uhHteLRww8NMSXD1fhHBcvHH-wF5f1ZMEj_A16s2jkHG-OKUuS5A080jncK9C-wN0CMotkV01D_GLw
5cJ9wXuNT4mHzwGnB496qKp01DoaF89j406uQlFI41SG6U_gzBGFfoQ80fj0BQ-0HpWkPOiPRgoVqI_w50ijc64jnFXwGJodI6eLP9zQSC0x00eYXzh96GjyHqGVI-8lPed_
FcqyHH0zRKSW4xc65YGtwZPTIPwL7UKYTiXYjRze7EE1pJvwGzqyyInwXXj04sef4o0qNJ008FCnSA7xJDf6x6jiHeMTt0KHJw060kGrq_idiWzx021E-UyYfcic0PR90lQ
ynEE_WCC0AQz4NaPP8X1h1WeTidDE0_hIFmjUGu3bjUH8G-zMMoPsGDbpQSPmDizn0fMhz9D18J2BEFQ8XDsup4EFtHWSE42eE9DxUZ5vQ6DH5cat4CEtWglnbBmVVA-
hkFVqah4SImaGQ90XAsPzcol-Egj2egYKRWALMqF8qqQHfQUYnoZHI7WHZayDxp9x86eqCqp2iid_i0h7mwG01k90yRj4fQZ8j54VXBK0d1NKUSHQ-NO_rQN3gzZPmD6g0a
-9gpJkaf_HADHeeM5vgh-6gzxREsxQ36D_Hj4heankX0QteD1EuJ_hj9Y3eIR8h3aGmke0jEBz-Kw32DB-2LnOzRL46CHnfQNBnWpfiwbYpQua3xD2_xH-c8XLrwIxc0T1mQ
xkWVUR3-Cg1VonwqHfCQ5sxQHuoYSXhznMioZmco5eQF02oC-Gi6Au0nEU4XcOkH-ZoHVeOC_4Kjx-u8EgjMMqLfpif3g724UeeQxxyHcd3nL7wHm7wHs9x0cIv-A-0-zBz
5A9-NDGRUwme4y9-S8jzBrt4-OC-TCp08Sp0Dc-HHg0jHnqyGL-FjhCZ9-iPH8cZ0gbzDE10HCenIGdEIViuvDpE8ZgkHrkz4j1-ofzBhMlk4RRj42kURA-elcfnoWGMkKxx
Z8SPHk3UCw8v4kR-Xie16zhV8Dxx7YRSNoaZRUAxDt_xIFyUIw_XRNjGsDjBPS3-BvsiQ8d5pMtyXBSuflDTJg2070G045mhPQ54Fz8aSxGNH1ouZRreC9-0SwuaG5reoskU
NmAsGzC52h-jNnxD-9S4duF-kEZXSERP_icwy4VPD2uDLWW1Gh0BXqk0L2Qksf7FX5PF1fjCxmTyogfQs-RR0Z3FmWi9riUHZ-J5gjPQ8-MPT_06fjy4NBtpJrCgme0oy-e
J9jeIKSaHLqmBaEfool8fPlxPEH25WicQRP1JWCZHJr2w3vxPZDUEPhx6sijKmB-MFzHvyD9mg0McelaMZ7oGypox1zsOSFPEqRvCP6HFNiicPzCLVZePORQzvyR0gPX-iT
4QmLZue85NDVoT_8RALudPsQaz8eilJQibvQCNUzw8too7wmpV-080EYALPlBJgmhW0JkSg266Gh4XAQpV8gb6DP8Cvnxog9uNkSoHz-h76jUwRxDVD-4YeiKzH0XcijiBwyu
bMYnHbkUeGNBde0iym4HyF_hEmm4m2HnnGg6HqQS8nROPlR48XvpJh2NH00i2gVBU195D_0X0FohQrRpIwVYN1h7shDoeUcfMef48q0JmH0DL2W4aRs4axQOQreo8Z3xDyh
PTvSozSuo2EoMqitD5y7gc_xoLa0Jm005EdyKY_wDjHHQ-Ph0TDY_rx4Ikf3D0q50g0ZgIyERdzm0gj1AptPG08-inIuF6I_sPR1149ImMS1ZgPcd5nDTPKEiWI_vxtfnx
Bc8jnEG86S4EP8Ih34Sn7caV00fz4AWHFJKEAAMAIgRRhxQSjAhADSPCBGEVYAI5oGgAIDGmBPgUSUZksIxxJRxBbHngUIGaKcGAIYtJbAgEhgmKCVIEACAKEAoApgxwFhi
AAKKKQOFSIA4KghfIhBiQFCEGAIEIQp45RjDDlgnUGAUAEbUEYiXyYgVCBGGUEGAIIiicobBLABCAFChhHKEaAsslQR6AhCggliiCCGgMakcQxQAQACHBGDCIMNkAYEIQ
BogCRBDFEABKAEE4SNw4ZRR0TIAgykaCAckMBcIAGxwggIAChAEQIE1dAIAIRRTWCH0rCXACnKZBEIKRYjQgCEihKFGEWIIIs4jQRAS1gnoH0EECagUEgIYA2AwAggmDQA
ICGAIEIGQwRgjAhDFAGEMQWAEIIAw8EwQFBBcAMACieMAA8z4AQhhEoIFVAEQKQs4QZAgCgAgBCCEAYYAMAYBiGahjgBBEB0CAETUQx8BiBiGABrADCEDEKYMEYIB4AS
CCAGiAcKECCeMCiAIoixAFijjCOAEOmYEQgEZZgBQR1CBEnIGiYcIcAQo4A1rIgEmCAAGKsAQIGAZMAi1BFqmBAMSaKIY4QAZwSIBgEvEKWECCQSGYRQAwARRGICCAOIGCCA
IMYAJIQwBwA1DGMMOQEAmmYBQRVxgg1NBPAAAIIMYIYZwggCghjKCIEMAUIhBZQIQgBgEhCAJOEUUIUACAIQARCHh1hAECEIAeYEQQoLQASgBfKCEhAgcEIBYBAwBQQAB
IBFKEEQEMgoBIYghdABngBCGKAickEAQCBUQFCgAgDHGACCAEIoYao1ghkKrnKDEECUEJkIQoQwRDAjFAAQIAAGVcEYIY4EHAD1BADBAKcoAEQgJ4AAx1EghgHKCECIMMwgg
QQAjBinnDSACGGCEJI4QIZxjCBgCBAUIG0gEEQg41kgUAlgAA
Consultando Banco de dados...
Música encontrada!
Titulo: Roudabout
Artista: Yes
aparece em: Amplified

```

Fonte: Elaborada pelo autor.

6 Conclusão

O objetivo deste projeto foi desenvolver um aplicativo modular que possa ser usado como ferramenta para o desenvolvimento de tecnologias web disponíveis atualmente, assim como o de propagar o conhecimento e facilitar a compreensão do assunto que envolve o tema de reconhecimento de sons.

O código de todo o projeto desenvolvido foi disponibilizado no *Github*, sob licença MIT, que garante permissão de cópia, reprodução, distribuição, publicação e venda para qualquer pessoa, reforçando o caráter público e comunitário. Para acesso ao código fonte do projeto e documentos utilizados durante o desenvolvimento acesse <<https://github.com/evandrododo/piratify>>.

Todas as etapas foram essenciais para a aplicação, e por se tratar de uma tecnologia em constante desenvolvimento a comunidade de programadores envolvidos contribuiu com as bibliotecas utilizadas tanto antes do início do projeto como durante, aplicando correções necessárias e modificações para melhoria de desempenho. O Resultado final foi uma aplicação de um modelo teórico simplificado e um protótipo funcional de reconhecimento de audio.

Referências

BARTSCH, M. A.; WAKEFIELD, G. H. Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on Multimedia*, v. 7, n. 1, p. 96–104, 2005. Disponível em: <<http://dblp.uni-trier.de/db/journals/tmm/tmm7.html\#BartschW05>>. Citado 2 vezes nas páginas 19 e 20.

BRITANAK, V.; YIP, P. C.; RAO, K. Chapter 3 - the karhunen-loève transform and optimal decorrelation. In: RAO, V. B. C. Y. (Ed.). *Discrete Cosine and Sine Transforms*. Oxford: Academic Press, 2006. p. 51 – 72. ISBN 978-0-12-373624-6. Disponível em: <<http://www.sciencedirect.com/science/article/pii/B9780123736246500059>>. Citado na página 18.

CANO, P. et al. A review of audio fingerprinting. *J. VLSI Signal Process. Syst.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 41, n. 3, p. 271–284, nov. 2005. ISSN 0922-5773. Disponível em: <<http://dx.doi.org/10.1007/s11265-005-4151-3>>. Citado 5 vezes nas páginas 16, 17, 18, 19 e 25.

HOLLAN, J.; HUTCHINS, E.; KIRSH, D. Distributed cognition: Toward a new foundation for human-computer interaction research. *ACM Trans. Comput.-Hum. Interact.*, ACM, New York, NY, USA, v. 7, n. 2, p. 174–196, jun. 2000. ISSN 1073-0516. Disponível em: <<http://doi.acm.org/10.1145/353485.353487>>. Citado na página 16.

JANG, D. et al. Pairwise boosted audio fingerprint. *IEEE Transactions on Information Forensics and Security*, v. 4, n. 4, p. 995–1004, 2009. Disponível em: <<http://dblp.uni-trier.de/db/journals/tifs/tifs4.html\#JangYLKK09>>. Citado na página 21.

KAPLAN, A. M.; HAENLEIN, M. Users of the world, unite! the challenges and opportunities of social media. *Business Horizons*, v. 53, n. 1, p. 59–68, 2010. Disponível em: <<http://EconPapers.repec.org/RePEc:eee:bushor:v:53:y:2010:i:1:p:59-68>>. Citado 2 vezes nas páginas 13 e 14.

LALINSKY, L. *How does Chromaprint work?* <https://oxygen.sk/2011/01/how-does-chromaprint-work/>, 2011. Citado 2 vezes nas páginas 21 e 22.

LERNER, J.; TIROLE, J. *The Simple Economics of Open Source*. [S.l.], 2000. (Working Paper Series, 7600). Disponível em: <<http://www.nber.org/papers/w7600>>. Citado na página 13.

MUSICBRAINZ. *MusicBrainz Database*. http://musicbrainz.org/doc/musicbrainz_database, 2015. Citado 2 vezes nas páginas 30 e 31.

SHEPARD, R. N. Circularity in judgments of relative pitch. *The Journal of the Acoustical Society of America*, v. 36, n. 12, p. 2346–2353, 1964. Disponível em: <<http://scitation.aip.org/content/asa/journal/jasa/36/12/10.1121/1.1919362>>. Citado na página 19.

WEBER, S. *The Success of Open Source*. Cambridge, MA, USA: Harvard University Press, 2004. ISBN 0674012925. Citado na página 14.