

PROJETO 2 RELATÓRIO FINAL

Universidade de Aveiro

Domingos Nunes (68710),
Dzianis Bartashevich (76537),
Francisco Cunha (76759),
Leonardo Oliveira (76725)



Versão 1.0

PROJETO 2 RELATÓRIO FINAL

Departamento de Eletrónica,
Telecomunicações e Informática
Universidade de Aveiro

Domingos Nunes (68710),
Dzianis Bartashevich (76537),
Francisco Cunha (76759),
Leonardo Oliveira (76725)

dfs@ua.pt,
bartashevich@ua.pt,
franciscomiguelcunha@ua.pt,
leonardooliveira@ua.pt

7 de Junho de 2015

Resumo

Com o objetivo de criar um pequeno órgão *Hammond* virtual, desenvolvemos uma aplicação *web* que permite criar músicas no formato Ring Tone Transfer Language (RTTTL) e interpretá-las com a sonoridade de um órgão deste tipo, com a possibilidade de aplicar efeitos.

Para tal, começámos por fazer o planeamento de todas as funcionalidades a implementar e, seguidamente, criámos cada um dos módulos pretendidos: versão móvel, versão *desktop*, base de dados, servidor e módulos relacionados com a geração de som.

Testámos e melhorámos a aplicação ao longo do tempo. O resultado foi uma aplicação que consegue gerar o conteúdo dinamicamente, guardando todos os ficheiros necessários antes, durante e depois de ser fechada.

Agradecimentos

Gostaríamos de agradecer aos nossos amigos e familiares; a todos os membros da comunidade da Universidade de Aveiro, em especial aos professores Diogo Gomes e João Barraca, que prontamente responderam a todas as nossas dúvidas na realização deste trabalho, e ao colega Ricardo Jesus, pelas suas sugestões.

Conteúdo

1	Introdução	1
2	Metodologia	2
3	Aplicação <i>web</i>	4
3.1	Aplicação móvel	4
3.1.1	Página Introdutória	4
3.1.2	Nova Música	4
3.1.3	Interpertação	4
3.1.4	PlaySong	8
3.1.5	About Us	8
3.2	Aplicação Desktop	8
3.2.1	Página inicial	10
3.2.2	Nova música	10
3.2.3	Nova Interpretação	10
3.2.4	Tocar música	10
3.2.5	Sobre nós	12
4	Servidor	14
4.1	index, novamusica, novainterpretacao, tocarmusica	14
4.2	createSong	14
4.3	createInterpretation	15
4.4	getWaveForm, getWaveFile	15
4.5	getNotes	15
4.6	listSongFiles	15
4.7	listNotes, listSongs	15
4.8	addVote, delVote	16
4.9	mobile	16
5	Som	17
5.1	Interpretador de pautas	17
5.2	Sintetizador	18
5.3	Processador de efeitos	18
5.3.1	Efeito nulo	19

5.3.2	Efeito eco	19
5.3.3	Efeito trémulo	19
5.3.4	Efeito distorção	19
5.3.5	Efeito percussão	19
5.3.6	Efeito coro	19
5.3.7	Efeito envelope	19
5.4	Imagem da Pauta	21
6	Base de dados	23
7	Validação	26
8	Considerações finais	28

Lista de Figuras

3.1	Página Introdutória da aplicação móvel.	5
3.2	Página de envio da música ao servidor.	6
3.3	Página de envio da interpretação ao servidor.	7
3.4	Página de reprodução das interpretações e mais algumas funcionalidades.	8
3.5	Página sobre o grupo responsável por este trabalho.	9
3.6	Página inicial da versão desktop da aplicação.	10
3.7	Página para adicionar uma nova música na versão desktop da aplicação.	11
3.8	Página para adicionar uma nova interpretação na versão desktop da aplicação.	11
3.9	Página para reproduzir músicas na versão <i>desktop</i> da aplicação. .	12
3.10	Página com informação dos elementos do grupo e agradecimentos.	13
5.1	Visualização gráfica através do terminal dos valores da lista utilizada para calcular os múltiplos da amplitude da onda no efeito percussão. No repositório, pode executar-se o ficheiro <code>som/decreasing_list_test.py</code>	20
5.2	Visualização gráfica através do terminal dos primeiros valores da lista utilizada para calcular os múltiplos da amplitude das amostras no efeito envelope. No repositório, pode executar-se o ficheiro <code>som/envelope_list_test.py</code>	21
5.3	Visualização gráfica das notas feita a partir da música The Simpsons.	22
6.1	Árvore da base de dados, com as tabelas da música (<i>musics</i>) e interpretação (<i>interpretation</i>).	24
7.1	Erro na criação de uma interpretação, depois da introdução de dados inválidos.	26
7.2	Página personalizada quando se tenta criar uma interpretação mas ainda não existe nenhuma música.	27

Capítulo 1

Introdução

Este documento contém uma abordagem detalhada dos módulos criados e das etapas de realização do projeto 2 de Laboratórios de Informática, de acordo com o enunciado disponibilizado na plataforma *moodle* da universidade [1].

Neste sentido, este documento está dividido em 8 capítulos. Depois desta introdução, no Capítulo 2 é explicada a metodologia seguida. No Capítulo 3 é apresentada a estrutura e funcionamento das páginas HyperText Markup Language (HTML) necessárias à interação com a aplicação, no Capítulo 4 são referidos os aspetos relacionados com o servidor, no Capítulo 5, são tratados os módulos necessários ao processamento e geração de ficheiros de som e no Capítulo 6 é apresentada a árvore da base de dados e métodos para o seu acesso. Já no Capítulo 7, estão presentes alguns pormenores sobre a validação de dados. Finalmente, no Capítulo 8 são apresentadas algumas considerações finais.

Capítulo 2

Metodologia

Para a realização deste projeto tivemos a seguinte ordem de trabalhos:

1. Análise do enunciado e consciencialização de todas as funcionalidades pretendidas;
2. Trabalho em separado em cada um dos componentes: implementação do mockup das páginas, servidor, base de dados e módulos relacionados com a produção de som;
3. Interligação entre os componentes, com especial atenção à interação entre a aplicação *desktop* e o servidor: páginas para criar músicas e interpretações;
4. Implementação da página para reproduzir músicas, ainda sem a funcionalidade dos votos;
5. Conclusão dos efeitos para o processador de efeitos;
6. Implementação dos pormenores finais, como a possibilidade de votar nas músicas e algumas validações para evitar erros de execução.

A distribuição das tarefas foi a seguinte:

- Domingos - HTML e Javascript da versão móvel da aplicação;
- Dzianis - Servidor, Javascript da versão móvel e da versão *desktop*;
- Francisco - Base de dados e métodos de acesso e modificação da base de dados;
- Leonardo - Módulos do som, HTML e Javascript da versão *desktop* da aplicação.

Para o desenvolvimento dos módulos ligados à geração de som, implementação do servidor e base de dados, foi utilizada a linguagem de programação

Python [2]. Recorreu-se ao módulo CherryPy [3], para o servidor, à ferramenta sqlite3 [4], para a base dados, e ao pacote wave [5], para o som.

Quanto às páginas *web*, basearam-se na *framework* JQuery Mobile [6], no caso da versão móvel, e no Bootstrap [7], para a versão *desktop*.

Foi utilizado um método para a deteção de dispositivos móveis da autoria de luchux [8], utilizador do GitHub. Este método foi útil para o servidor poder decidir que versão enviar quando uma página *web* lhe é pedida: *desktop* ou *mobile*.

Ao longo deste documento serão descritas as implementações realizadas, sendo que, para obter informação mais detalhada sobre cada assunto, poderão ser consultados os comentários efetuados ao longo do código fonte.

Capítulo 3

Aplicação *web*

3.1 Aplicação móvel

A versão móvel da aplicação contém cinco páginas: introdutória, nova música, interpretação, playSong e About Us.

3.1.1 Página Introdutória

Nesta página (ver Figura 3.1) faz-se uma introdução da aplicação móvel com uma ligação para entrar na página "nova música". Esta página é a única que não contém as barras fixas superior e inferior de navegação: na superior encontra-se o nome da página actual e um botão para regressar à página introdutória com atualização da aplicação móvel (lado direito). Finalmente, na barra inferior, temos disponíveis as ligações de interesse para as restantes páginas da aplicação móvel. (ver Figura 3.2, Figura 3.3, Figura 3.4 e Figura 3.5).

3.1.2 Nova Música

Nesta secção, o utilizador pode escrever as suas pautas na caixa de texto criada para o efeito e enviar para o servidor através de um botão (ver Figura 3.2). Quando o processamento é finalizado é enviado uma mensagem ao utilizador de acordo com o sucesso ou o insucesso do mesmo.

3.1.3 Interpretação

Esta página corresponde ao último passo que o utilizador tem de dar para a criação do ficheiro audio. Para isso, ele tem de escolher a música, registo, efeito e o nome para o ficheiro, como podemos ver na figura Figura 3.3. Como na página "Nova Música", o utilizador vai receber mensagens de aviso após o processamento do servidor.

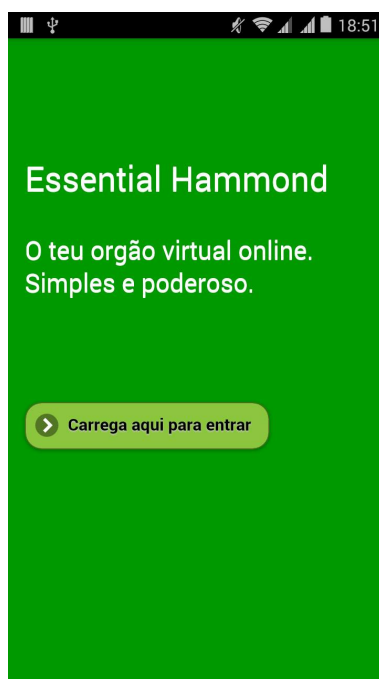


Figura 3.1: Página Introdutória da aplicação móvel.



Figura 3.2: Página de envio da música ao servidor.



Figura 3.3: Página de envio da interpretação ao servidor.

3.1.4 PlaySong

Inicialmente, nesta página, o utilizador tem disponível um *pop-up* para escolher a música. Ao pressionar no botão "Procurar", a app móvel irá encontrar as interpretações que a musica anteriormente escolhida está associada e serão imediatamente visíveis. Cada interpretação irá conter um *slidedown* individual com várias funcionalidades: contagem de gostos e não gostos, botões de gosto e não gosto, um botão "imagem" em que a imagem da pauta será visualizada através de um *pop-up* e, finalmente, a reprodução audio da interpretação em questão (ver figura Figura 3.4).

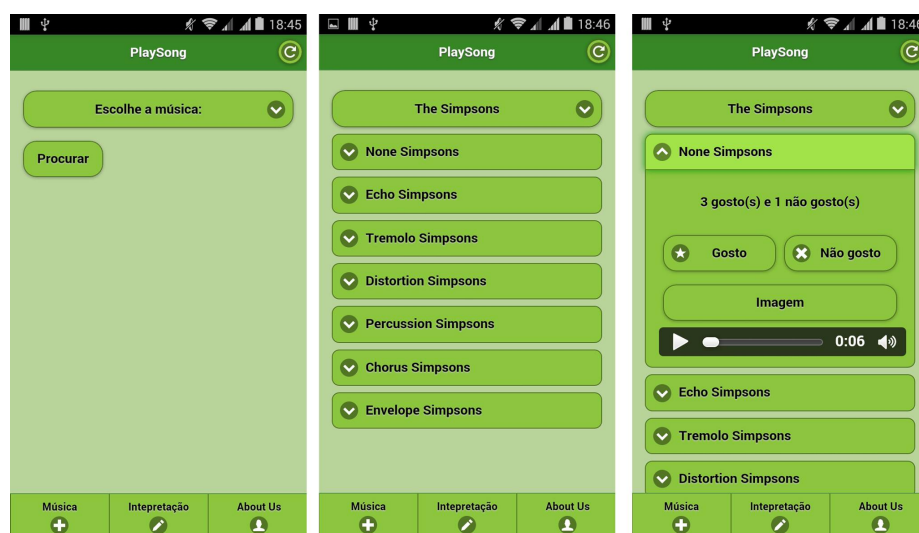


Figura 3.4: Página de reprodução das interpretações e mais algumas funcionalidades.

3.1.5 About Us

Nesta página, serão visualizados todos os alunos responsáveis pela a criação desta aplicação. Ao carregar num aluno abre-se um *pop-up* contendo mais informações sobre o aluno, como a cidade, idade, curso e n^o mecanográfico. (ver figura Figura 3.5).

3.2 Aplicação Desktop

A versão desktop da aplicação contém cinco páginas: inicial, nova música, nova interpretação, tocar música e sobre nós.

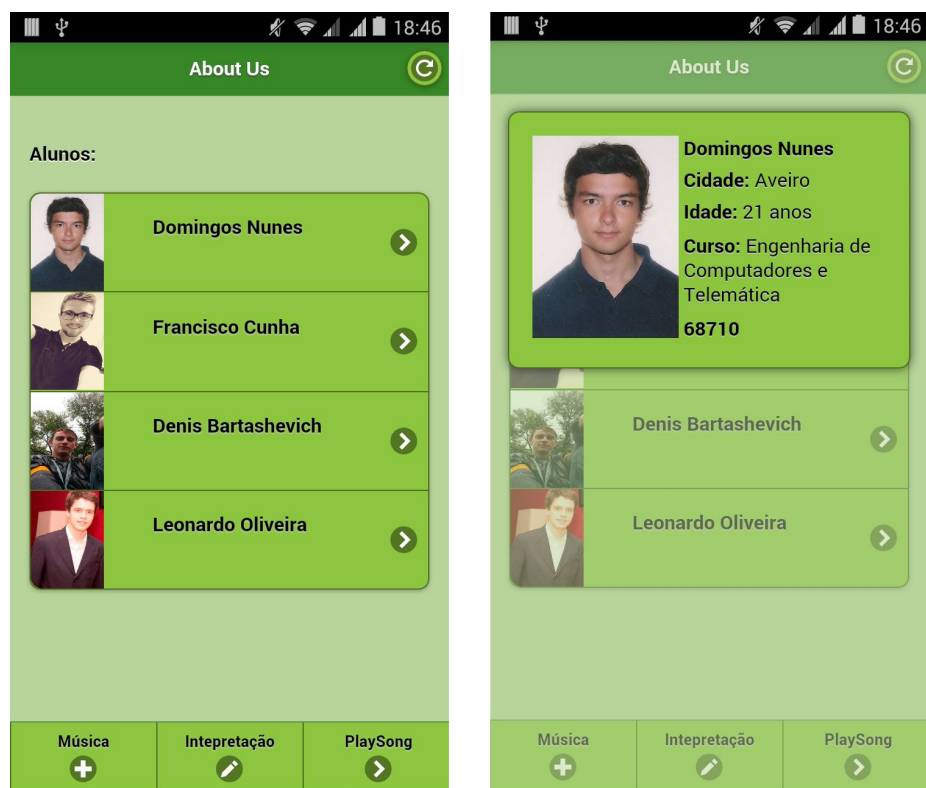


Figura 3.5: Página sobre o grupo responsável por este trabalho.

3.2.1 Página inicial

Nesta página faz-se uma breve apresentação da aplicação, disponibilizando-se ligações de interesse para as restantes páginas (ver Figura 3.6).

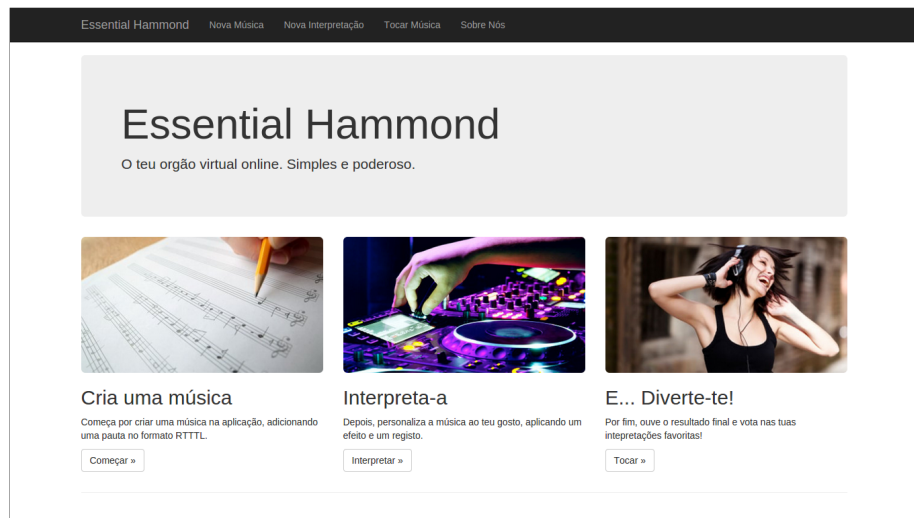


Figura 3.6: Página inicial da versão desktop da aplicação.

3.2.2 Nova música

Aqui, o utilizador pode adicionar as suas pautas no formato RTTTL (ver Figura 3.7). Para tal, introduz-se a pauta na caixa de texto disponibilizada e carrega-se em enviar música, sendo mostrada uma mensagem de sucesso à direita do botão quando o servidor termina o processamento. É utilizado o serviço `/createSong` (para enviar a música para o servidor).

3.2.3 Nova Interpretação

Tem um funcionamento semelhante à página anterior, com a existência de mais opções para escolher: nome desejado para a interpretação, música a escolher de entre as existentes, efeito e registo. Mostra igualmente mensagens de aviso, sendo que na Figura 3.8 é possível ver a mensagem intermédia "A Enviar...". São utilizados os serviços `/createInterpretation` (para enviar a interpretação para o servidor) e `/listNotes` (para listar as músicas existentes).

3.2.4 Tocar música

Esta página dispõe de um menu lateral onde se pode seleccionar a música pretendida. Para cada música, é mostrada a imagem da pauta e as respetivas

Essential Hammond Nova Música Nova interpretação Tocar Música Sobre Nós

Adicionar nova música

Pauta The Simpsons : d=4, o=5, b=160 : c.6, e6, f#6, 8a6, g.6, e6, c6, 8a, 8f#, 8f#, 2g, 8p, 8p, 8f#, 8f#, 8g, a#., 8c6, 8c6, 8c6, i

Enviar música

Música enviada com sucesso!

Copyright © 2015. Todos os direitos reservados. Desenvolvido em [Bootstrap](#). Template original: [Shop Item](#).

Figura 3.7: Página para adicionar uma nova música na versão desktop da aplicação.

Essential Hammond Nova Música Nova interpretação Tocar Música Sobre Nós

Adicionar nova interpretação

Nome pretendido The Simpsons Rock Style

Música The Simpsons

Efeito Percussão

Registo 888888888

Enviar interpretação

A enviar...

Copyright © 2015. Todos os direitos reservados. Desenvolvido em [Bootstrap](#). Template original: [Shop Item](#).

Figura 3.8: Página para adicionar uma nova interpretação na versão desktop da aplicação.

interpretações existentes, com a possibilidade de efetuar votos (ver Figura 3.9). São utilizados os serviços `/addVote` (para enviar um voto positivo), `/delVote` (para enviar um voto negativo), `/getWaveForm` (para obter o ficheiro de som), `/getWaveFile` (para obter a imagem da pauta), `/listNotes` (para listas as músicas existentes) e `/listSongs` (para listar as interpretações existentes).

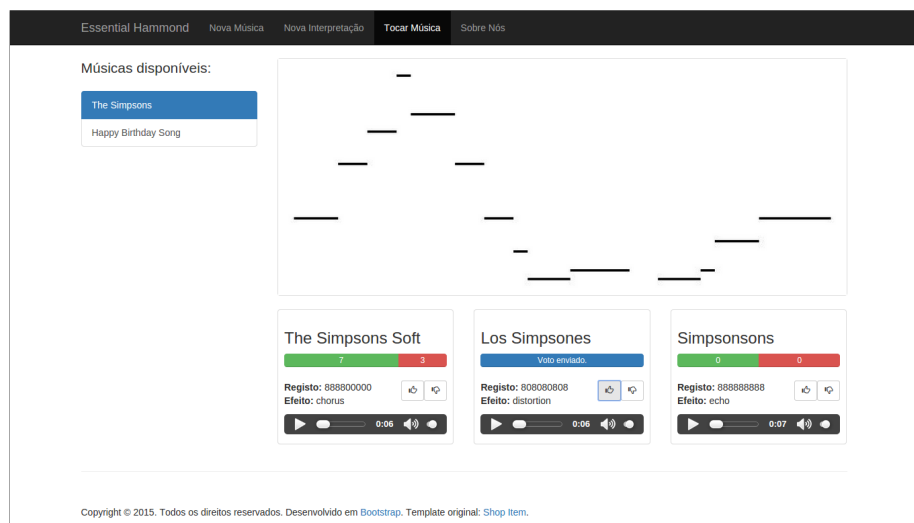


Figura 3.9: Página para reproduzir músicas na versão *desktop* da aplicação.

3.2.5 Sobre nós

Disponibiliza informação sobre os quatro elementos do grupo, apresentando também os agradecimentos a quem colaborou com sugestões no nosso projeto (ver Figura 3.10).

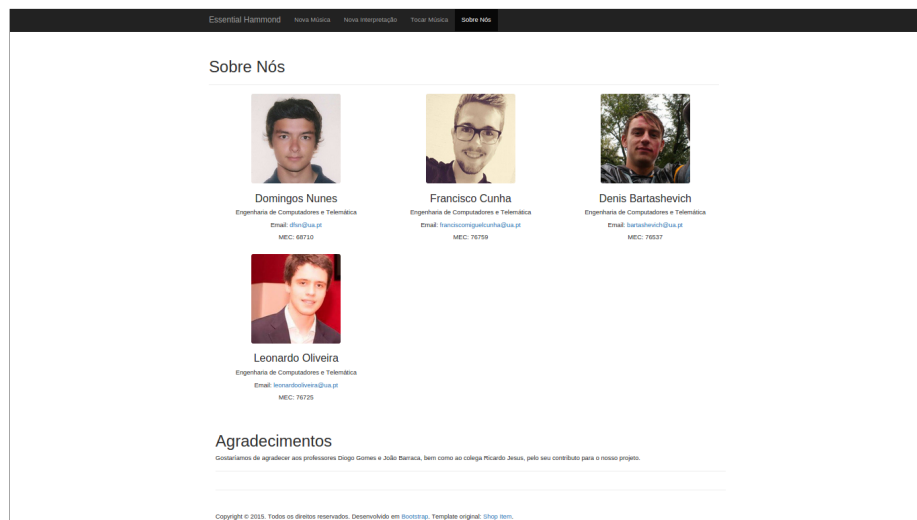


Figura 3.10: Página com informação dos elementos do grupo e agradecimentos.

Capítulo 4

Servidor

O programa `server.py` foi construído com o módulo CherryPy [3]. Fizemos as seguintes importações: `cherrypy`, `sqlite3` e `os.path`. As restantes importações são programas construídos pelos membros do grupo: `interpreter`, `synthesizer`, `effects_processor` e `database`.

O `server.py` possui uma única class (`Root`) onde se encontram os seguintes módulos criados: `mobile`, `addVote`, `delVote`, `index`, `novainterpretacao`, `tocarmusica`, `sobrenos`, `createSong`, `createInterpretation`, `getWaveForm`, `getWaveFile`, `getNotes`, `listSongFiles`, `listNotes` e `listSongs`.

4.1 `index`, `novamusica`, `novainterpretacao`, `tocarmusica`

Verifica se o website está a ser acedido através de um dispositivo móvel ou computador. Caso seja móvel irá redirecionar sempre para o `index` da versão *mobile*. Caso contrário, devolve a página pedida na versão *desktop*.

4.2 `createSong`

Recebe como argumentos o nome e a pauta, verificando se estão minimamente válidos (nome não pode ser vazio, a pauta não pode estar vazia - quando o javascript envia notas vazias, o módulo recebe-as no formato `undefined:undefined`). Após a verificação é feita decodificação para *String* dos parâmetros recebidos. Depois de converter é criada a imagem com o nome igual ao id das notas na tabela `musics` da base de dados. A criação de imagem também serve de método de verificação pois, se ocorrer algum erro na criação, uma exceção é lançada e o programa envia uma mensagem de erro. Caso a imagem seja criada com sucesso, os dados são enviados para base de dados e é devolvida uma mensagem de sucesso.

4.3 createInterpretation

Recebe como argumentos o registo, o id da pauta, o efeito e o nome, fazendo a verificação (registo tem de ser numérico, tamanho do registo só pode ser 9, o registo não pode conter o algarismo 9, nome não pode ser vazio, efeito não pode ser vazio, id não pode ser vazio, id tem de ser numérico). Antes de criar o ficheiro .wav é necessário ir buscar e juntar o nome e as notas do id fornecido como argumento. Caso na criação do ficheiro .wav ocorra uma exceção, o programa imprime uma mensagem de erro. Se o ficheiro for criado com sucesso, a informação sobre a interpretação (argumentos do módulo) irá ser guardada na base de dados e irá ser devolvida uma mensagem de sucesso.

4.4 getWaveForm, getWaveFile

Recebe como argumento o id da música/interpretação, que depois irá redirecionar para o ficheiro .jpg/.wav. Caso o ficheiro não exista será lançada a exceção *404NotFound*.

4.5 getNotes

Recebe como argumento o id da tabela musics, devolvendo as notas na forma de *String*.

4.6 listSongFiles

Recebe como argumento o id da tabela musics, onde irá buscar todas as interpretações feitas sobre essa música, retornando na forma de JavaScript Object Notation (JSON).

4.7 listNotes, listSongs

Não recebe nenhum argumento, devolvendo em formato JSON todas as tabelas de musics (tabela das notas) / interpretations (tabela das interpretações), respetivamente.

Formato do json devolvido (listNotes):

```
[
  {
    "notes": "d=4,o=5,b=160:c.6, e6, f#6, 8a6, g.6, e6, c6, 8a, 8f#,
             8f#, 8f#, 2g, 8p, 8p, 8f#, 8f#, 8f#, 8g, a#., 8c6, 8c6, 8c6,
             c6",
    "id": 1,
    "name": "The Simpsons"
  }
]
```

```
]
```

Formato do json devolvido (listSongs) :

```
[
  {
    "name": "The Simpsons Rock Style",
    "id_music": 1,
    "downvotes": 10,
    "effects": "distortion",
    "registration": 888800000,
    "upvotes": 2,
    "id": 1
  }
]
```

4.8 addVote, delVote

Recebe como argumento o id da interpretação onde irá adicionar, na tabela interpretations, uma unidade ao campo upvote / downvote, respetivamente. Caso a interpretação não exista ou a operação não ocorra por outro motivo, devolve uma mensagem de erro.

4.9 mobile

(Para testes) Força a aparecer a versão *mobile* do site, mesmo que o utilizador esteja a aceder a partir de um computador.

Capítulo 5

Som

Os módulos ligados ao processamento e geração de ficheiros de som seguiram os moldes do que foi sugerido no enunciado do projeto. Foi criado um interpretador de pautas, um sintetizador e um processador de efeitos. Criámos também um método para a criação de imagens com a representação das notas das músicas.

5.1 Interpretador de pautas

O interpretador recebe uma pauta no formato RTTTL e devolve uma lista de pares duração-frequência, cada par correspondendo à nota e sua respetiva duração. Tomemos como exemplo a seguinte pauta:

"Barbie girl : d=4, o=5, b=125 : 8g#, 8e, 8g#, 8c#6, a, p, 8f#, 8d#, 8f#, 8b, g#, 8f#, 8e, p, 8e, 8c#, f#, c#, p, 8f#, 8e, g#, f#"

Neste caso, o interpretador devolve uma lista com 23 pares, com a seguinte estrutura:

```
[{'freq': 830, 'time': 0.24}, {'freq': 659, 'time': 0.24}, {'freq': 830, 'time': 0.24}, {'freq': 1108, 'time': 0.24} ... ]
```

Internamente, o interpretador começa por ignorar a primeira parte da pauta, correspondente ao nome. De seguida, analisa os valores dos parâmetros de referência, d, o e b. Caso algum não esteja definido, é aplicado o valor padrão (d=4, o=6 e b=63). Por fim, extrai a parte correspondente às notas e percorre nota a nota, tendo a vírgula como referência para as separar. Para determinar a frequência da nota para uma dada oitava, foi criada uma *lookup table*, sendo devolvida a frequência quando é inserido como parâmetro a seguinte expressão: $[12 * \text{oitava} + \text{tom}]$ - com a oitava a variar entre 0 e 7 e o tom entre 0 e 11. O

0 correspondente ao dó e o 11 ao si. No final do cálculo da frequência e tempo de cada nota, o par é adicionado à lista, que no final é devolvida.

5.2 Sintetizador

O sintetizador recebe como argumentos os pares vindos do interpretador e um registo, devolvendo uma nova lista com a frequência (principal) e respetivas amostras ao processador de efeitos. Inicialmente são criadas as seguintes variáveis:

- sounds - lista vazia, que irá ser a devolvida no final;
- freqs - lista de tamanho 9, que irá conter os 9 frequências para cada nota;
- ratio - lista de tamanho 9 com os múltiplos para o cálculo das 9 frequências para cada nota, de acordo com os osciladores: $[1/2, 2/3, 1, 2, 3, 4, 5, 6, 8]$;
- amplitudes - lista de tamanho 9 que irá conter as amplitudes das frequências para cada nota, de acordo com os valores definidos no registo.
- rate - inteiro para a frequência de amostragem, que é sempre 44100.

De seguida, o sintetizador percorre a lista de pares recebido do interpretador, alterando a lista de frequências para cada nota. Produz as amostras com a duração definida, correspondendo à soma das 9 componentes sinusoidais. Estas têm a amplitude e frequência previamente calculadas e guardadas nas listas referidas acima. No final de cada nota ser processada, a frequência principal e amostras são adicionadas à lista sounds, devolvida no final. A lista fica com a seguinte estrutura:

```
[{"freq": 830, "samples": [0, 11775, 18804, 19193, 14747, 9339, ...],  
  {"freq": 659, "samples": [...], ...}]
```

Os cuidados com a possível existência de *clipping* não são tomados nesta altura, pelo ue existirá um método para normalizar as amostras mais à frente, no processador de efeitos.

5.3 Processador de efeitos

O processador de efeitos recebe a lista de sons vinda do sintetizador e o efeito pretendido, tendo como função gerar o ficheiro de som da música. Nas secções seguintes irá ser explicado como é aplicado cada um dos efeitos.

Depois do efeito pretendido ser aplicado, as amostras são normalizadas, para poderem estar contidas no intervalo de resolução (-32768 to 32767), sendo posteriormente empacotadas e utilizadas na geração do ficheiro wav.

5.3.1 Efeito nulo

A partir da lista de sons fornecida, extrai a lista de amostras de cada som e junta-os numa única lista, para poder devolvê-la no final.

5.3.2 Efeito eco

Aplica inicialmente o efeito nulo e, seguidamente, percorre a lista de amostras, somando a cada uma um múltiplo (inferior a 1, para atenuar) da amostra correspondente a 0,1 segundos atrás e outro múltiplo (inferior ao anterior, para atenuar ainda mais, pois é um eco mais atrasado) da amostra 0,2 segundos atrás. Deste modo, são introduzidos dois ecos, com atrasos de 0,1 e 0,2 segundos.

5.3.3 Efeito trémulo

Aplica inicialmente o efeito nulo e, seguidamente, percorre a lista de amostras, adicionando a cada amostra uma outra amostra de uma sinusoidal de frequência 20Hz. O resultado é uma música com oscilações rápidas de amplitude, dando a ideia de vibração.

5.3.4 Efeito distorção

Aplica inicialmente o efeito nulo e, seguidamente, percorre a lista de amostras, elevando ao quadrado cada amostra.

5.3.5 Efeito percussão

Tem como objetivo de no início da música ou depois de uma pausa sobrepor uma harmónica, que se vai anulando ao longo do tempo. Para tal, começa por aplicar no primeiro conjunto de amostras essa harmónica (início da música) e, para as notas seguintes, volta a aplicar uma harmónica, caso a frequência principal das amostras da nota imediatamente anterior seja zero (pausa).

A modulação da amplitude de cada harmónica ao longo do tempo é baseada numa lista previamente criada (ver Figura 5.1), na qual se pode aceder a um índice entre 0 e 200. Para cada índice, vem um múltiplo (entre 0 e 1) pelo qual se deve multiplicar a amplitude da onda.

5.3.6 Efeito coro

Para cada nota, percorre a lista de amostras, adicionando 3 sinusoidais com +20, +30 e +50 Hz que a frequência principal, tendo como efeito um conjunto de sons de fundo que acompanham a música principal.

5.3.7 Efeito envelope

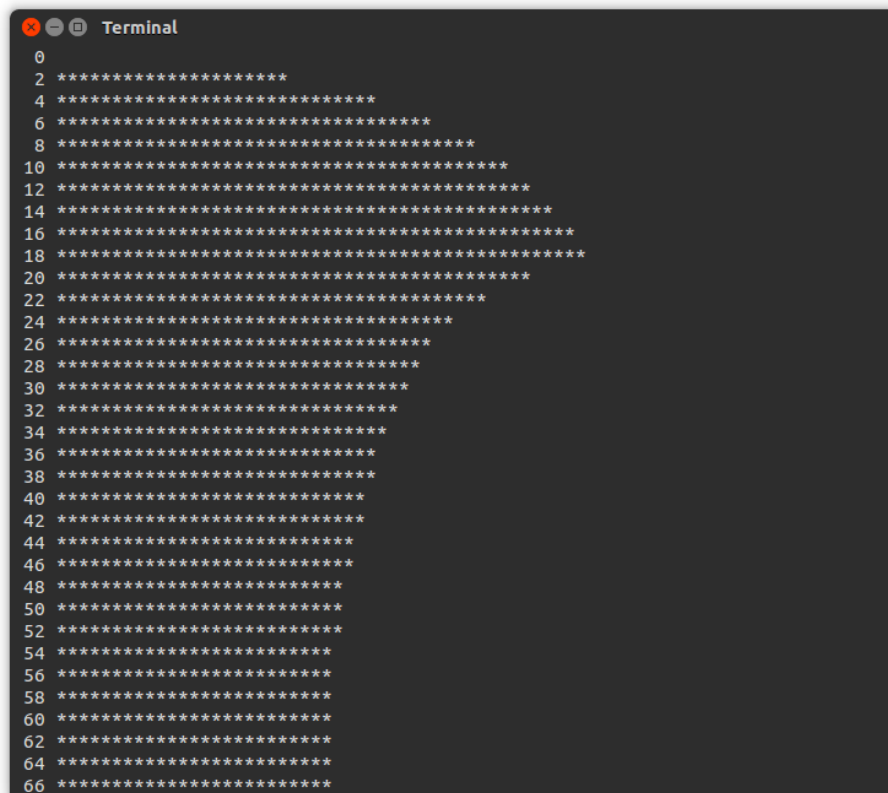
Para cada nota, percorre a lista de amostras, moldando a amplitude total da nota de acordo com uma lista previamente criada (ver Figura 5.2), na qual se

```
Terminal
0 *****
8 *****
16 *****
24 *****
32 *****
40 *****
48 *****
56 *****
64 *****
72 *****
80 *****
88 *****
96 *****
104 *****
112 *****
120 *****
128 *****
136 *****
144 *****
152 *****
160 *****
168 *****
176 *****
184 *****
192 *****
200 *****

-----
(program exited with code: 0)
Press return to continue
█
```

Figura 5.1: Visualização gráfica através do terminal dos valores da lista utilizada para calcular os múltiplos da amplitude da onda no efeito percussão. No repositório, pode executar-se o ficheiro som/decreasing_list_test.py.

pode aceder a um índice entre 0 e 200. Para cada índice, vem um múltiplo (entre 0 e 1) pelo qual se deve multiplicar a amostra.



```
Terminal
0
2 *****
4 *****
6 *****
8 *****
10 *****
12 *****
14 *****
16 *****
18 *****
20 *****
22 *****
24 *****
26 *****
28 *****
30 *****
32 *****
34 *****
36 *****
38 *****
40 *****
42 *****
44 *****
46 *****
48 *****
50 *****
52 *****
54 *****
56 *****
58 *****
60 *****
62 *****
64 *****
66 *****
```

Figura 5.2: Visualização gráfica através do terminal dos primeiros valores da lista utilizada para calcular os múltiplos da amplitude das amostras no efeito envelope. No repositório, pode executar-se o ficheiro `som/envelope_list_test.py`.

5.4 Imagem da Pauta

O método que cria as imagens chama-se `create_image` e recebe como argumento os pares vindos do interpretador e uma string com indicação do nome pretendido para o ficheiro.

Para tal, é utilizada a biblioteca PIL [9], que tem a utilidade de gerar o ficheiro JPEG final.

O método começa por criar uma nova imagem toda a branco, com dimensões calculadas a partir do número de notas e do espetro de frequências possíveis. Foi escolhido o modo de representação de cor Black and White (BW), visto

pretender-se escrever as notas a preto em fundo branco. Deste modo, a imagem gerada fica também um pouco mais leve.

Seguidamente, o método percorre a lista de pares. O posicionamento da nota na imagem é calculado baseando-se na frequência, para a altura, e na duração, para o comprimento. São alterados para preto os pixels correspondentes à zona onde a nota deve ficar desenhada. Caso a nota seja uma pausa (frequência zero), esta não é representada, visto tratar-se de uma pausa.

No final do processo a imagem é recortada, dado que na maioria das vezes o espectro de frequências e, por consequência, a altura da imagem, nunca é utilizada na sua totalidade. O resultado final pode ser observado na Figura 5.3, que representa as notas da música The Simpsons.



Figura 5.3: Visualização gráfica das notas feita a partir da música The Simpsons.

Capítulo 6

Base de dados

A base de dados associada a este projeto são duas tabelas - **musics**, **interpretations** - que estão interligadas entre si e as quais servirão para armazenar informação acerca dos conteúdos.

A tabela **musics** conterá informação com o **name** das músicas criadas e as **notes** (pautas) que lhes são correspondentes, sendo que a estes dois campos há um **id** que os identifica.

Por outro lado, a tabela **interpretation** terá campos como:

- **id_music**, que estará associado à tabela **musics**;
- **(registration)** (registos), que servirá para a codificação do registo;
- **effects** (efeitos), servindo para guardar informação sobre e os efeitos utilizados na música;
- **upvotes** e **downvotes** guardarão respectivamente o número de votos positivos e negativos.

A Figura 6.1 apresenta o esquema da árvore da base de dados.

Dentro do ficheiro que acede à base de dados, há uma condição que verifica se se encontra criada a base de dados. Caso não exista, é o próprio programa que a cria, através do ficheiro *create.txt*.

```
if not os.path.isfile('songs.db'):

    if not os.path.isfile('create.txt'):
        print 'We need a database to run the program. We don\'t have one,
              so we need a "create.txt" file to create the library. Copy a
              valid "create.txt" file to this folder and reopen the program.'
        exit(1)

    db = sql.connect('songs.db')

    createDBfile = open('create.txt', 'r')
```

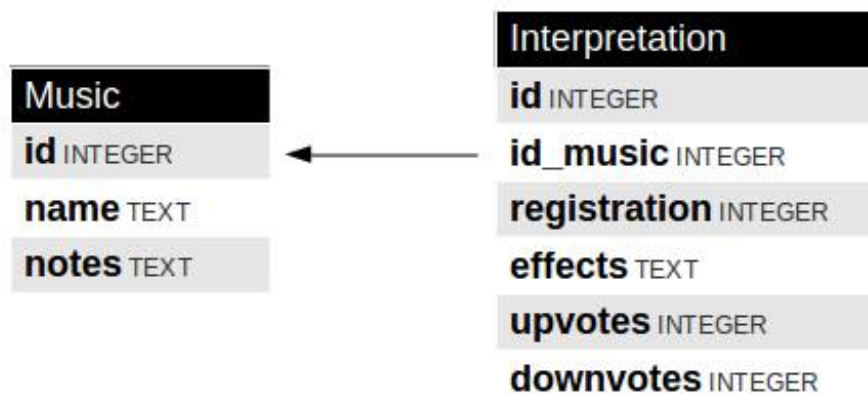


Figura 6.1: Árvore da base de dados, com as tabelas da música (*musics*) e interpretação (*interpretation*).

```

for line in createDBfile:
    db.execute(line)

db.commit()
createDBfile.close()
db.close()
  
```

Dentro do servidor, é utilizado um conjunto de métodos que permitem efetuar operações na base de dados.

```

def create_song(name, notes):
    command = 'INSERT INTO musics(name, notes) VALUES(' + name + ', ' +
              + notes + ')"'
    db.execute(command)
    db.commit()
  
```

A função acima descrita permite criar uma nova música na tabela das músicas, sendo que a cada nova música adicionada irá haver uma **id** que lhe é atribuída, facilitando desta forma a identificação da música mais facilmente quando é necessário fazer alterações. Para criar uma nova interpretação seguem-se os mesmos padrões do exemplo anterior.

Tal como é possível adicionar novas músicas, também se pode visualizar o conteúdo de uma tabela, neste caso, o conteúdo da **interpretations**, através do exemplo seguinte:

```
def get_all_notes():
    command = 'SELECT * FROM musics'
    result = db.execute(command)
    rows = result.fetchall()
    d = []
    for row in rows:
        name = {"id":row[0], "name":row[1], "notes":row[2]}
        d.append(name)

    return d
```

Para ser possível atualizar o número de votos, neste caso positivos, sendo o mesmo método aplicado no número de gostos negativos, recorreu-se ao seguinte código. Esta função vai mostrar os votos positivos de uma determinada interpretação, sendo o valor guardado numa variável onde será incrementado em mais uma unidade. Após isso executa-se um novo comando e atualiza-se o número de gostos positivos para o novo valor.

```
def add_upvotes(ID):
    command = 'SELECT upvotes FROM interpretations WHERE id = ' + str(ID)
    result = db.execute(command)
    result = result.fetchone()
    result = result[0] + 1
    command = 'UPDATE interpretations SET upvotes = ' + str(result) + ' '
               'WHERE id = ' + str(ID)
    db.execute(command)
    db.commit()
```

Como se pode verificar pelo código abaixo descrito, esta função vai tentar buscar todos os **id**'s da tabela interpretações, guardando-os numa lista. Como apenas o último id é pretendido, acede-se à posição `len(x)-1`. Caso a lista esteja vazia, é lançada uma exceção, que é tratada.

```
def last_id_interpretations():
    try:
        result = db.execute("SELECT id FROM interpretations")
        rows = result.fetchall()
        x = []
        for row in rows:
            x.append(row[0])

        return x[len(x)-1]+1
    except IndexError, e:
        return 1
```

Capítulo 7

Validação

Ao longo do projeto foram tidos em conta alguns cuidados de validação, a fim de evitar erros desnecessários. Alguns desses cuidados foram referidos atrás, como o tratamento de exceções e a validação dos valores enviados para o servidor provenientes das caixas de texto das páginas *web*. Relativamente a este último caso, quando o servidor deteta alguma ilegalidade, devolve uma mensagem de erro, que é interpretada pela aplicação *web* e anunciada ao utilizador através de uma mensagem de aviso (ver Figura 7.1).

A screenshot of a web form. At the top, there is a label 'Registo' followed by a text input field containing the value '934959392'. Below the input field is a blue button with the text 'Enviar interpretação'. To the right of the button, there is a red rectangular box containing the text 'Ocorreu um erro. Por favor, insere valores válidos.'

Figura 7.1: Erro na criação de uma interpretação, depois da introdução de dados inválidos.

Outro exemplo de validação ocorre quando ainda não existem músicas adicionadas e, portanto, não é possível interpretar nem tocar músicas. Nesse caso, é apresentada uma página de aviso personalizada (ver Figura 7.2).

Foram feitos outros pequenos testes ao longo do desenvolvimento do projeto que aqui não serão referidos, dentro dos quais o ficheiro `som/tests.py`, que contém diversos testes unitários feitos aos módulos de geração de som.

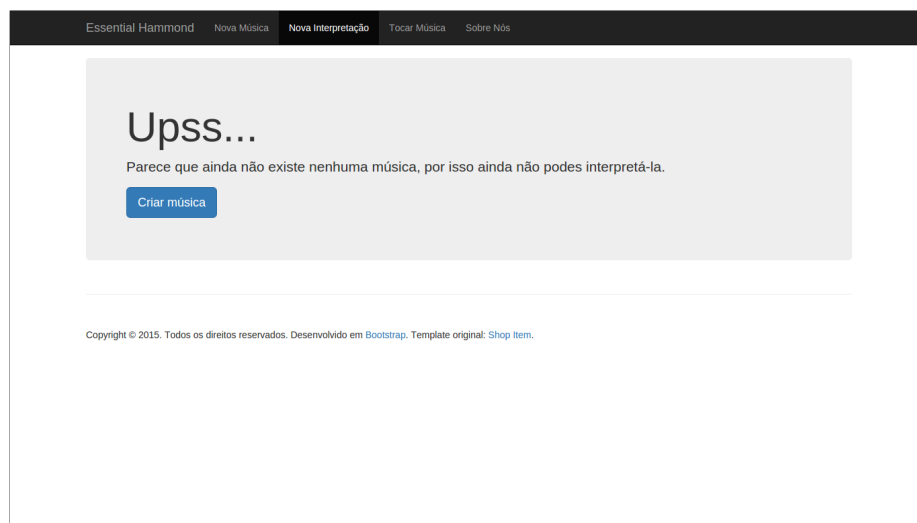


Figura 7.2: Página personalizada quando se tenta criar uma interpretação mas ainda não existe nenhuma música.

Capítulo 8

Considerações finais

O desenvolvimento do projeto decorreu de acordo com o planeado, tendo, no entanto, havido algumas dificuldades na gestão do tempo para o cumprimento das tarefas.

Foram atingidos os objetivos pretendidos: aplicação com versão *desktop* e *mobile*; possibilidade de adicionar músicas e interpretações, com efeitos e registos; e reprodução de músicas, com a possibilidade de votar.

Acrónimos

HTML HyperText Markup Language

RTTTL Ring Tone Transfer Language

JSON JavaScript Object Notation

BW Black and White

Bibliografia

- [1] *Laboratórios de informática*, [Online; acedido em Junho de 2015]. endereço: <http://elearning.ua.pt/course/view.php?id=3470>.
- [2] *Python*, [Online; acedido em Junho de 2015]. endereço: <https://www.python.org/>.
- [3] *Cherrypy*, [Online; acedido em Junho de 2015]. endereço: <http://www.cherrypy.org/>.
- [4] *Sqlite*, [Online; acedido em Junho de 2015]. endereço: <http://www.sqlite.org/>.
- [5] *Wave*, [Online; acedido em Junho de 2015]. endereço: <https://docs.python.org/2/library/wave.html>.
- [6] *Jquery mobile*, [Online; acedido em Junho de 2015]. endereço: <https://jquerymobile.com/>.
- [7] *Bootstrap*, [Online; acedido em Junho de 2015]. endereço: <http://getbootstrap.com/>.
- [8] *Luchux*, [Online; acedido em Junho de 2015]. endereço: <https://github.com/luchux/cherrypy-mobile-detection>.
- [9] *Pil*, [Online; acedido em Junho de 2015]. endereço: <https://pillow.readthedocs.org/>.