PROJETO 2 (Proj2)

Universidade de Aveiro

António Macedo, João Oliveira, João Tomás Simões, Martim Neves (Grupo 3)

MIECT

PROJETO 2 (Proj2)

Laboratórios de Informática (Turma 4)

Universidade de Aveiro

António Macedo, João Oliveira, João Tomás Simões, Martim Neves (Grupo 3)
(66425) amrm@ua.pt, (87874) joaomsoliveira@ua.pt
(88930) jtsimoes@ua.pt, (88904) martimfneves@ua.pt

Junho 2018

Agradecimentos

Deixamos aqui um agradecimento ao professor João Paulo Barraca e ao professor Vitor Cunha pela enorme disponibilidade de ambos em ajudarem-nos neste nosso trabalho e pelas dicas valiosas que nos recomendaram a implementar para que o nosso código ficasse ainda melhor.

Sem a vossa ajuda não seria possível a conclusão deste projeto. Mais uma vez, obrigado por nos serem tão prestáveis e por esclarecerem as nossas inúmeras dúvidas. Esperamos que esta nossa aplicação esteja do vosso agrado.

Conteúdo

Capítulo 1

Introdução

No âmbito da unidade curricular de Laboratórios de Informática (LABI), foi-nos proposto a realização de um Projeto 2 (PROJ2) cujo objetivo foi a criação de um sistema que permite que sejam geradas músicas através de trechos de outras músicas.

Para isso foi feito uso de uma interface web, através da qual é feito todo o contacto com o utilizador. Além disso, foi também utilizada uma Aplicação Web (WEB APP) que permite o fluxo de informação entre as várias páginas e os seus múltiplos constituintes. De seguida foi utilizada uma Base de Dados (DATA BASE) para armazenar todas as informações relativas a cada música. Por fim foi criado um gerador de músicas que, passando-lhe uma pauta como argumento, gera a música contida nessa pauta.

Assim, neste documento, irá perceber-se qual foi a nossa maneira de pensar e de implentar o código, assim como explicá-lo e testá-lo. Para isso, este documento está dividido em 7 capítulos.

Depois desta introdução virá o Capítulo 2, onde apresentamos o nosso programa divido em blocos de código (funções) e em baixo de cada um explicamos o nosso pensamento enquanto o implementávamos e explicamos também qual a sua função. No Capítulo 3 são apresentados alguns testes feitos por experimentação que demonstram a funcionalidade do programa. De seguida, no Capítulo 4, serão discutidas as dificuldades na elaboração do código, bem como os bugs conhecidos e outros tópicos considerados relevantes. Finalmente, no Capítulo 5 são apresentadas as conclusões deste trabalho.

Capítulo 2

Explicação dos vários blocos de código (funções)

2.1 *Interface Web*

Para a criação da interface web foi necessário criar três páginas, cada uma delas construída com HyperText Markup Language (HTML), JavaScript (JS) e Cascading Style Sheets (CSS) que, como já foi referido, vão ser o único meio de contacto entre o utilizador e o sistema.

Na primeira página são listadas todas as músicas por ordem alfabética crescente, bem como um reprodutor de áudio que permite que todas as músicas sejam reproduzidas. Existem ainda botões de voto (positivo ou negativo), através dos quais os vários utilizadores podem expressar a sua opinião relativamente a cada música.

```
<h2>Song Files:</h2>

<div id="bar">
    <div id="likes"></div>
    <div id="dislikes"></div>
</div>

<ul id="mySongList">

</ul>

<p>Go back to the <a href="/">main page</a>.</p>

<audio id="myPlayer" controls>
    <source id="myAudioSource" src=""></source>
    Your browser does not support the audio format.
</audio>
<!-- <button type="button" class="repeat" id="myRepeat" onclick="repeatAudio()">Repeat OFF</button> -->

<!-- Scripts -->
<script>
    mySongList.onclick = function(e) {
        e.preventDefault();

        var elm = e.target;
        var player = document.getElementById("myPlayer");

        var source = document.getElementById("myAudioSource");
        source.src = elm.getAttribute("data-value");

        player.load();
        player.play();
    };
</script>
```

Figura 2.1: Página 1

Na primeira página é criado o reprodutor de áudio e é chamada uma função desenvolvida em ***Python*** que vai listar todas as músicas existentes.

```
$("#mySongList").ready(function(){

    //Function to open json file
    $.getJSON("/list?type=songs", function(data) {
    //add Samples

    Sli ="";
    for(i=0;i<data.length;i++){
        Sli='<li><a href="" data-value="/audio/' + data[i].name + '.wav">' + data[i].name + '    </a>';
        Sli=Sli +'<button class="btn btn-success btn-sm" type="button"';
        Sli = Sli+ 'onclick="';
        Sli=Sli + "location.href='/vote?id=" + data[i].id + "&user=Tester&points=1'" +';alert('+ "'You liked this song!'"+');"';
        Sli=Sli + '><i class="fas fa-thumbs-up"> </i></button>  ';

        Sli=Sli +'<button class="btn btn-danger btn-sm" type="button"';
        Sli = Sli+ 'onclick="';
        Sli=Sli + "location.href='/vote?id=" + data[i].id + "&user=Tester&points=-1'" +';alert('+ "'You disliked this song!'"+');"';
        Sli=Sli + '><i class="fas fa-thumbs-down"> </i></button>    ';
        $("#mySongList").append(Sli);
    }

    });
});
```

Figura 2.2: Função para listar músicas

Esta função abre o ficheiro JavaScript Object Notation (JSON) e cria uma ***string*** à qual acrescenta o nome de todas as músicas existentes, fazendo uso de um ciclo ***for*** para percorrer o ficheiro. Além disso são também criados dois botões para o utilizador poder votar positiva ou negativamente na musica.

Na segunda página são listados todos os trechos existentes, os quais podem ser utilizados mais tarde para a criação de músicas. Tal como na primeira página, existe também um reprodutor para que seja possível a reprodução de todas as músicas. Além disso, existe um botão que permite que o utilizador envie um trecho seu para mais tarde poder ser utilizado na composição de músicas.

```
<h2>Sample Files:</h2>
<ul id="mySampleList">

</ul>

<form action="../upload" method="post" enctype="multipart/form-data">
    <input type="file" name="myFile" accept="audio/wav"></input>
    <input type="submit" value="Upload"></input>
</form>

<p>Go back to the <a href="/">main page</a>.</p>

<audio id="myPlayer" controls>
    <source id="myAudioSource" src=""></source>
    Your browser does not support the audio format.
</audio>
<!-- <button type="button" class="repeat" id="myRepeat" onclick="repeatAudio()">Repeat OFF</button> -->

<!-- Scripts -->
<script>
    mySampleList.onclick = function(e) {
        e.preventDefault();

        var elm = e.target;
        var player = document.getElementById("myPlayer");

        var source = document.getElementById("myAudioSource");
        source.src = elm.getAttribute("data-value");

        player.load();
        player.play();
    };
</script>
```

Figura 2.3: Página 2

Na página 2 é criado um botão que permite que o utilizador envie um ***sample*** seu para usar na composição de músicas. Para além disso é chamada outra função desenvolvida em ***Python*** que, tal como a função da página 1, lista o conteúdo do fiheiro JSON, só que desta vez retorna uma lista de ***samples***.

```
$("#mySampleList").ready(function(){

    //Function to open json file
    $.getJSON("/list?type=samples", function(data) {
    //add Samples

    Sli ="";
    for(i=0;i<data.length;i++){
        $("#mySampleList").append('<li><a href="" data-value="../audio/'+ data[i].name +'.wav">'+ data[i].name +'</a></li>');
    }

    });
});
```

Figura 2.4: Função para listar ***samples***

Tal como referido anteriormente, esta função faz o mesmo que a função da página 1 (abrir o ficheiro JSON e criar uma ***string*** à qual vão sendo adicionados os vários ***samples*** à medida que se percorre o ficheiro).

Na terceira página existe uma tabela que permite que o utilizador escolha um trecho e um efeito a aplicar a esse trecho. Ao longo de vinte tempos, o utilizador vai escolher quando quer que cada trecho seja utilizado e, carregando num botão, será enviada para a WEB APP uma pauta correspondente às escolhas do utilizador, pauta essa que será processada e transformada numa música.

```
<table id="SongTable" align=center class="myTable">
        <tr>
            <th>AUDIO</th>
            <th>1</th>
            <th>2</th>
            <th>3</th>
            <th>4</th>
            <th>5</th>
            <th>6</th>
            <th>7</th>
            <th>8</th>
            <th>9</th>
            <th>10</th>
            <th>11</th>
            <th>12</th>
            <th>13</th>
            <th>14</th>
            <th>15</th>
            <th>16</th>
            <th>17</th>
            <th>18</th>
            <th>19</th>
            <th>20</th>
            <th>EFFECTS</th>
        </tr>
        <tr>
            <th><select id="mySelect1"><option value="none">Choose audio</option></select></th>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.1"  onclick="playAudio('myCheck1.1', 'mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.2"  onclick="playAudio('myCheck1.2', 'mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.3"  onclick="playAudio('myCheck1.3', 'mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.4"  onclick="playAudio('myCheck1.4', 'mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.5"  onclick="playAudio('myCheck1.5', 'mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.6"  onclick="playAudio('myCheck1.6', 'mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.7"  onclick="playAudio('myCheck1.7', 'mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.8"  onclick="playAudio('myCheck1.8', 'mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.9"  onclick="playAudio('myCheck1.9', 'mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.10" onclick="playAudio('myCheck1.10','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.11" onclick="playAudio('myCheck1.11','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.12" onclick="playAudio('myCheck1.12','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.13" onclick="playAudio('myCheck1.13','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.14" onclick="playAudio('myCheck1.14','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.15" onclick="playAudio('myCheck1.15','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.16" onclick="playAudio('myCheck1.16','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.17" onclick="playAudio('myCheck1.17','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.18" onclick="playAudio('myCheck1.18','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.19" onclick="playAudio('myCheck1.19','mySelect1')"><label></label></td>
            <td class="checkbox_wrapper"><input type="checkbox" id="myCheck1.20" onclick="playAudio('myCheck1.20','mySelect1')"><label></label></td>
            <td><select id="myEffect1"><option value="none">Choose effect</option></select></td>
```

Figura 2.5: Página 3

Na terceira e última página é criada uma tabela com várias ***checkboxes***, tabela essa que vai servir para o utilizador definir que trechos quer que toquem, em que momento e com que efeito, gerando assim uma pauta. Nesta página existem também um botão ***rec***, para guardar a pauta gerada e criar a música, e um botão ***reset***, que deseleciona todas as ***checkboxes***, trechos e efeitos, tal como mostrado na figura seguinte.

```
<center>
<a href="#" title="" class="add-another">Add another line</a>
</center>

<br>

<form class="text-center">
    <button type="button" onclick="resetAll()">Reset</button>
    <input type="text" id="inName" placeholder="Song Name" required />
    <input type="submit" id="recButton" value="Create Now" />
</form>
```

Figura 2.6: Página 3

Existe ainda uma opção para adicionar mais uma linha, caso o utilizador queira uma música que use mais ***samples***. Para isso foi desenvolvida em JS a seguinte função:

```
counter = 5;
$("#SongTable").ready(function(){

    //Function to open json file
    $.getJSON("/list?type=", function(data) {
    //add Samples
    for(j=1;j<=counter;j++){
        for(i=0;i<data.length;i++){
            $("#mySelect"+j).append("<option value = /audio/"+ data[i].name +".wav>"+ data[i].name +"</option>");
        }
    }
    //add Effects
    for(j=1;j<=counter;j++){
        $("#myEffect"+j).append("<option value ='fadein'>Fade in</option>");
        $("#myEffect"+j).append("<option value ='fadeout'>Fade out</option>");
    }

    });
});
```

Figura 2.7: Função para acrescentar linha

Esta função faz uso de um ciclo ***for*** para criar uma nova entrada e adicionar todos os trechos e efeitos possíveis de serem utilizados.

```
$("#recButton").click(function(){
    var bpm=120;
    var samples=[];
    var effects=[];
    var music=[];
    var name ="";

    for(j=1;j<=20;j++){
        var q=0;
        var col=[];

        for (i=1;i<=5;i++){
            //obtém os valores das colunas
            if (document.getElementById("myCheck"+i+"."+j).checked == true){
                col[q]=i-1;
                q++;
            }
        }
        music[j-1]=col;
    }
    var c=0;
    for(i=0;i<counter;i++){
        //obtém os samples e os effects
        var s = document.getElementById("mySelect"+(i+1));
        var str = s.options[s.selectedIndex].text;
        if(s.options[s.selectedIndex].value != "none"){
            samples[c]= str  ;
            c++;
            //effects
            var temp=[];
            var e = document.getElementById("myEffect"+(i+1));
            var str = e.options[e.selectedIndex].value;
            if(str != "none"){
                temp[i]= str;
            }else{
                temp[i]= [] ;
            }
            effects[i]=temp[i];
        }
    }

    var pauta={"bpm": bpm,"samples": samples,"effects": effects, "music": music};
    var jpauta= JSON.stringify(pauta)
    names=document.getElementById("inName").value;

    res = names.split(" ");;
    if (res[0] != ""){
        $.post( "/put", {sheet : jpauta ,name : names})
        .done(function( data ) {
            alert('The song "' + names + '" has been created!');
        });
```

Figura 2.8: Botão para gerar música

O botão ***rec*** é descrito pela função acima apresentada. Através de vários ciclos ***for*** é possível saber quais os ***samples*** e efeitos selecionados e os tempos em que cada ***sample*** é reproduzido. Esses elementos constituem a pauta, pauta essa que é depois convertida para JSON.Analisando o nome dado à música que está prestes a ser criada, se o nome for válido, a música é criada. Caso contrário, deverá ser introduzido um novo nome.

2.2 *Aplicação Web*

Nesta secção foi desenvolvida a WEB APP. A WEB APP consiste essencialmente num programa em ***Python*** que serve conteúdos estáticos. É através da WEB APP que são executadas todas as funcionalidades deste projeto, nomeadamente:

1. /**list?type=songs**: devolve um objeto JSON com todas as músicas já criadas e a respetiva informação de cada música, como por exemplo data de criação, nome, identificador, etc.

2. /**list?type=samples**: muito semelhante ao anterior no entanto, em vez de devolver um objeto JSON com todas as músicas e as suas caraterísticas, o objeto JSON é referente aos trechos existentes e às suas caraterísticas.

3. /**get?id=identificador**: permite identificar uma música ou um trecho através do seu identificador.

4. /**put**: permite o envio de uma pauta, juntamente com o nome da música, para que seja criada uma nova música apartir da pauta.

5. /**vote?id=identificador&user=uid&points=1**: permite que um determinado utilizador (autenticado pelo próprio XCOA ao iniciar a WEB APP) vote numa determinada música, sendo que o voto pode ser positivo ou negativo.

A declaração das funções são muito semelhantes entre si e o código interno [1] das mesmas também é facilmente compreensível, mas mesmo assim vamos explicar algum mais complexo nos capítulos seguintes.

2.3 *Persistência*

Nesta secção do trabalho o objetivo foi a criação de uma DATA BASE relacional que contivesse toda a informação acerca dos trechos e músicas existentes, sendo que apenas a informação se encontra na DATA BASE pois os ficheiros WAVEform audio file format (WAVE) que contêm os trechos e as músicas estão guardados no sistema de ficheiros da WEB APP. Além disso foi também necessária a criação de métodos para ler e escrever informação na DATA BASE.

	id	name	length	date	uses	votes	author	type	songid
1	1	demo	42	20-12-2018	0	0	user1	Song	{null}
2	2	example	82	25-12-2018	0	-1	user1	Song	{null}
3	3	claves	0	25-12-2018	0	0	user1	Sample	{null}
4	4	conga	0	25-12-2018	0	1	user1	Sample	{null}
5	5	conga2	0	25-12-2018	0	0	user1	Sample	{null}
6	6	crash	0	25-12-2018	0	-1	user1	Sample	{null}
7	7	gun	0	25-12-2018	0	0	user1	Sample	{null}
8	8	hat	0	25-12-2018	0	0	user1	Sample	{null}

Figura 2.9: Tabela com informação dos ***samples***

```
def list(self, type):
    #print(type)
    db = sqlite3.connect("SongsDataBase.db",check_same_thread=False)
    data =[] # Cria o dicionário

    if (type == 'songs'):
        result = db.execute('SELECT * From Songs WHERE type = "Song"')
        #print("Songs:")
    elif (type == 'samples'):
        result = db.execute('SELECT * From Songs WHERE type = "Sample"')
        #print("Samples:")
    else:
        #print("All:")
        result = db.execute("SELECT * From Songs")

    rows = result.fetchall()
    for row in rows:
        linha = {}
        # Linhas para o ficheiro JSON
        linha['name'] = row[1]
        linha['id'] = row[0]
        linha['length'] = row[2]
        linha['date'] = row[3]
        linha['uses'] = row[4]
        linha['votes'] = row[5]
        linha['author'] = row[6]
        linha['type'] = row[7]
        linha['songid'] = row[8]
        data.append(linha) # Adiciona a linha no dicionário

    data = json.dumps(data)
    #print(data)
    return data
```

Figura 2.10: Exemplo de leitura de dados da DATA BASE

O código acima mostra como se podem ler os dados contidos na DATA BASE. Primeiro é estabelecida ligação à DATA BASE. Depois, selecionando apenas as músicas, ***samples*** ou todo o conteúdo existente na DATA BASE, o resultado vai ser impresso. Após a impressão dos resultados, todas as informações correspondentes aos dados impressos são passadas para o ficheiro JSON.

```python
def upload(self, myFile):
    db = sqlite3.connect("SongsDataBase.db",check_same_thread=False)
    fo = open(os.getcwd()+ "/audio/" + myFile.filename, "wb")
    while True:
        data = myFile.file.read(8192)
        if not data:
            break
        fo.write(data)
    fo.close()

    fo=wave.open("audio/"+myFile.filename,"rb")
    frames=fo.getnframes()
    rate=fo.getframerate()
    MD5 = hashlib.md5()
    ret = []
    while fo.tell() < fo.getnframes():
        decoded = struct.unpack("hhl", fo.readframes(1))
        ret.append(decoded)
    MD5.update(ret)
    id = MD5.hexdigest()
    id = id[0:16]
    print()
    print()
    print()
    print(id)
    print()
    print()
    print()

    name=str(myFile.filename).split(".")
    duration=frames/float(rate)
    duration=float("{0:.2f}".format(duration))
    now = datetime.datetime.now()
    db.execute("INSERT INTO Songs (name,length,date,uses,votes,author,type,songid) VALUES (?,?,?,?,?,?,?,?)",(name[0],str(duration),str(now.day)+"-"+str(now.month)+"-"+str(now.year),0,0,str(Root.user),"Sample",""))
    db.commit()
    db.close()

    raise cherrypy.HTTPRedirect("/html/page2.html", 301)
```

Figura 2.11: Exemplo de escrita de dados na DATA BASE

Neste exemplo de escrita de dados na DATA BASE, tal como no exemplo acima, a primeira coisa a fazer é estebelecer ligação à DATA BASE. De seguida é mostrado o código para o botão de ***upload***. É criado um ***id*** baseado na conteúdo da música e de seguida as informações referentes a essa música são escritas na DATA BASE.

2.4 *Gerador de Músicas*

Nesta parte do projeto foi desenvolvido o gerador de músicas. Este gerador permite que o utilizador lhe mande uma pauta com dados de uma música e o gerador vai transformar essa pauta em música, de acordo com os dados nela presentes. Caso a geração seja bem sucedida, o ficheiro WAVE vai ser guardado no sistema de ficheiros e a informação relativa à música gerada será armazenada na DATA BASE, sendo que o identificador da música é gerado de acordo com o seu conteúdo.

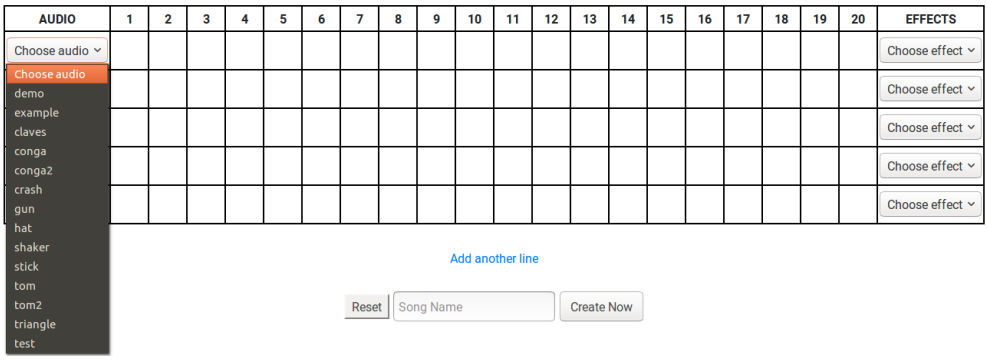

Figura 2.12: Interface do Gerador de Músicas

```
def put(self, sheet, name):

    data=json.loads(sheet)

    bpm=str(data['bpm'])
    print(bpm)

    samples=str(data['samples'])
    samples=samples.replace('[','')
    samples=samples.replace(']','')
    samples=samples.replace('"','')
    samples=samples.replace(" ",'')
    samples=samples.split(',')
    print(samples)


    effects= str(data['effects'])
    effects=effects.replace('[','')
    effects=effects.replace(']','')
    effects=effects.replace('"','')
    effects=effects.split(',')
    print(effects)


    music = str(data['music'])
    music=music.split(']')
    pauta=[]
    for row in music:
        row=row.replace('[','')
        row=row.replace(',','')
        print(row)
        row=row.split(' ')
        pauta.append(row)

    print(pauta)
    data=[]
    for row in pauta:
        trim=0
        for s in row:
            if (s!=""):
                fo=wave.open("audio/"+samples[int(s)]+".wav","rb")
                length = fo.getnframes()
                for i in range(0,length):
                    waveData = fo.readframes(1)
                    waveData = struct.unpack("hhl", waveData)
                    trim =trim+wave

        data.append(trim)

    wv = wave.open("audio/"+name+".wav", "w")
    wvData=[]

    for v in data:
        wvData += pack("h", int(v))

    wv.writeframes(bytearray(wvData))
    wv.close()
```

Figura 2.13: Código para criar música a partir da pauta

Neste excerto de código é mostrado o processo de criação de músicas através da pauta. São selecionados todos os trechos e efeitos usados na criação e são adicionados no mesmo ficheiro WAVE nos tempos selecionados pelo utilizador.

Capítulo 3

Testes

Ao longo da escrita do nosso código fomos implementando e ao mesmo tempo testando algumas funções e verificámos apenas um bug em todo o nosso código.

O bug existe no que toca aos votos. Teoricamente, o utilizador só podia dar um *like* ou um *dislike*. Quando o utilizador clicava no botão de *like*, o mesmo botão ficava com o atributo *disable* e o utilizador apenas poderia clicar no botão *dislike* e vice-versa. Ao fazer alguns testes, verificámos que isso não acontece. E, por falta de tempo, não conseguimos corrigir este erro mas ponderámos ainda um pouco e chegámos à conclusão que muito provavelmente está relacionado com a DATA BASE e não com o JS nem com o HTML.

É de notar também que o tempo não permitiu adicionar efeitos às músicas no criador de sons (página 3). O dicionário JSON ainda foi iniciado mas não conseguimos concluir o código Python para os aplicar às músicas geradas pela página 3. Não tendo tempo para concluir os módulos obrigatórios, certamente que não houve tempo para realizar também o módulo bónus intitulado "Gerador de Imagens", embora com muita pena nossa visto que não parecia complicado.

Outro pequeno detalhe que verificámos foi que se o utilizador selecionasse uma música muito longa na tabela de criar sons (página 3), ela ficaria a tocar até ao fim e caso o utilizador selecionasse mais sons, tornava-se irritante e muito barulhento. A solução que encontrámos foi colocar todas as músicas a tocar apenas por 5 segundos seguido de um *fade out* quando o utilizador marca a *checkbox* correspondente.

De resto, a aplicação está a executar normalmente e como pedido no guião, não havendo mais erros graves a notar.

Capítulo 4

Discussão

Tivemos algumas dificuldades nomeadamente aquando do envio da WEB APP para o XCOA, visto que alterou imensos caminhos que tivemos de retificar posteriormente. Entraram em conflito também alguns problemas na base de dados mas foi tudo corrigido, mais uma vez graças à ajuda e dicas dos docentes.

Uma coisa que foi notada por todos os membros deste grupo foi que o projeto era um pouco extenso para o tempo dado. Como se verifica pela data do primeiro *commit*, mesmo começando o projeto quando o guião foi lançado (como foi o caso do nosso grupo), achámos que faltou algum tempo para adicionar efeitos, corrigir erros, adicionar mais funcionalidades e principalmente modificar mais os estilos CSS, melhorando assim o design da nossa interface.

Mas, mesmo assim, verificámos que conseguimos gerir bem o tempo, graças aos *commits* diários e ao empenho árduo na realização deste PROJ2. Estamos assim orgulhosos e contentes com o resultado final deste nosso projeto.

Capítulo 5

Conclusão

Para a realização deste projeto foi utilizada a pasta ***proj2*** no diretório principal do XCOA do grupo do João Tomás Simões (JT) e do Martim Neves (MN) ***(labi-t4g7@xcoa.av.it.pt)*** e o projeto no CodeUA com o identificador ***labi2018-p2-g3*** que pode ser consultado em `http://code.ua.pt/projects/labi2018-p2-g3`.

Achamos que o nosso código está bastante completo e funcional visto que resultou de horas de trabalho intensivas até ao último dia da entrega do mesmo, quer na escrita do código quer nos testes do mesmo para nos certificarmos que não haveria qualquer problema, quer na utilização dos trechos para compor novas músicas, quer em todas as outras funcionalidades que incluem gravação dos dados na base de dados, contagem de votos, etc.

Gostámos de realizar este trabalho pois abordou muitos dos conceitos e linguagens de programação essenciais que vamos utilizar, não só durante o curso, mas também ao longo da nossa vida profissinal e serviu para que pudessemos melhorar as nossas aptidões ao nível desses mesmo conceitos e linguagens.

Para além disso foi também uma boa maneira de consolidarmos conhecimentos por nós próprios e para não acumularmos matéria, permitindo uma maior facilidade para a realização do teste teórico.

Capítulo 6

Contribuições dos autores

Este trabalho resultou da interação e colaboração de todos os membros do grupo, quer a nível de escrita de código, quer a nível criativo. A troca de ideias entre todos os membros permitiu a elaboração de um trabalho mais completo e de maior qualidade, pelo que todos os membros foram igualmente importantes e contribuiram para a realização deste projeto.

Todas as pessoas deste grupo (António Macedo (AM), JT, João Oliveira (JO) e MN) trabalharam de igual forma para a realização deste projeto, tendo cada um realizado aproximadamente a mesma percentagem de trabalho (25%).

Capítulo 7

Glossário

LABI Laboratórios de Informática

AM António Macedo

JO João Oliveira

JT João Tomás Simões

MN Martim Neves

PROJ2 Projeto 2

JSON JavaScript Object Notation

WEB APP Aplicação Web

DATA BASE Base de Dados

CSS Cascading Style Sheets

HTML HyperText Markup Language

JS JavaScript

WAVE WAVEform audio file format

Bibliografia

[1] João Paulo Barraca & Diogo Gomes, André Zúquete, João Manuel Rodrigues, António Adrego da Rocha, Tomás Oliveira e Silva, Sílvia Rodrigues, Óscar Pereira, António Neves, Pétia Georgieva, Vitor Cunha, José Duarte. ***Guiões Práticos da disciplina***. Disponíveis em: `http://elearning.ua.pt/`. Última vez acedido: 19/04/2018.