

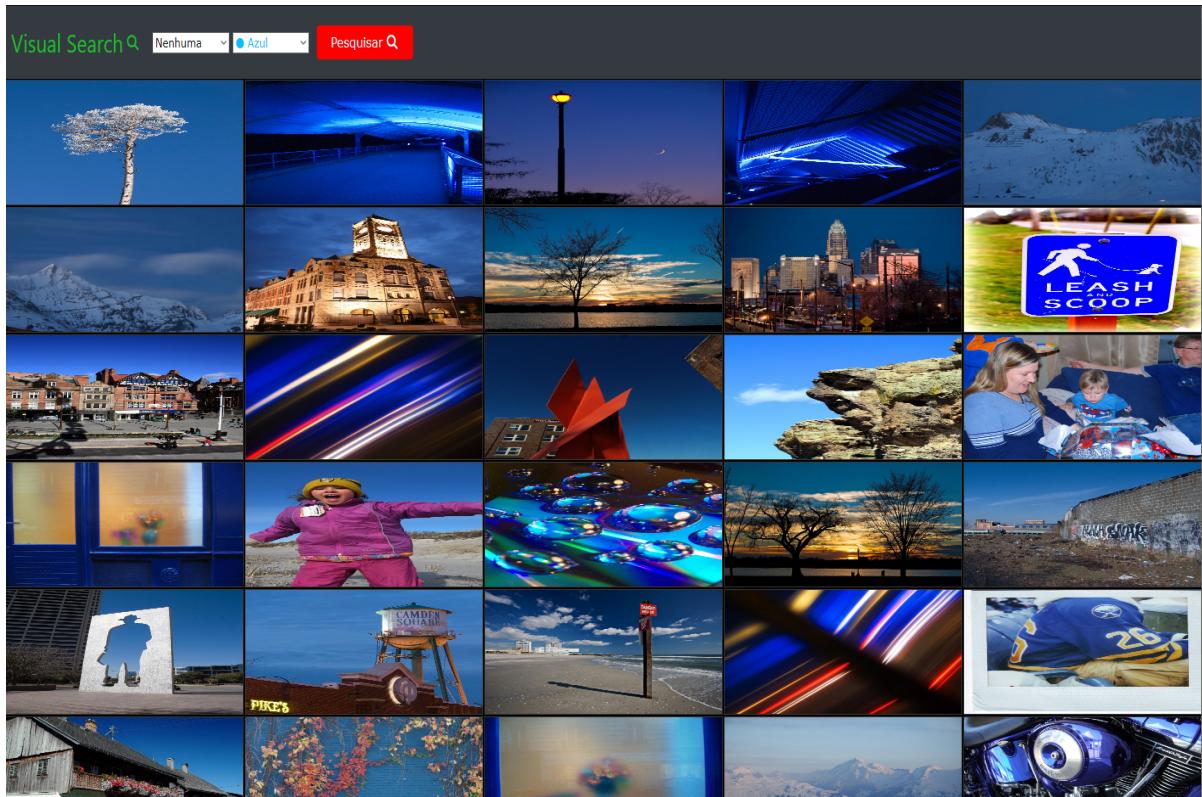


Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Produção de Conteúdos Multimédia - 2022/2023 SI

Projeto - Pesquisa de Fotos



Docente Rui Jesus

Realizado por :
Pedro Silva 48965
Diogo Santos 48626

Conteúdo

1	Introdução	I
2	Desenvolvimento da Aplicação	I
2.1	Pesquisa por Palavras-Chave	I
2.2	Visualização das Imagens	II
2.3	Lista de cores	III
2.4	Pesquisa por Cor	IV
3	Descrição da Aplicação Final	VIII
4	Discussão das Questões mais Relevantes	VIII
5	Conclusões	X

Listas de Figuras

1	Seleção da categoria	I
2	Método "searchKeywords"	I
3	Método "gridView"	II
4	Seleção da cor	III
5	Lista de Cores - HTML	III
6	Cálculo do histograma de cores	IV
7	Código da computação das imagens	V
8	Método "imageProcessed"	V
9	Método "createXMLColodatabadeLS"1/2	VI
10	Método "insertInColorArray"	VI
11	Método "createXMLColodatabadeLS"2/2	VII
12	Método "searchColor"	VII
13	Aplicação Final	VIII
14	Código para escolha de cor sem categoria	IX
15	Exemplo-1	IX

1 Introdução

Este projeto tem como objetivo o desenvolvimento de uma aplicação multimédia em HTML5, incluindo API's em JavaScript, para pesquisa e visualização de fotos digitais de uma coleção de fotografias. É nos disponibilizado um conjunto de fotografias, uma base de dados que contém a informação de cada uma delas e o código base para o desenvolvimento desta aplicação. Foi desenvolvida a possibilidade de pesquisa por palavras-chave e por cor(utilizando o histograma de 12 cores que substitui a pesquisa por cor utilizando a cor dominante fornecida no ficheiro XML), na pesquisa por cor uma lista de cores para o utilizador escolher qual deseja e por fim a exibição das imagens está organizada numa grelha de imagens.

2 Desenvolvimento da Aplicação

2.1 Pesquisa por Palavras-Chave

Começamos por desenvolver a pesquisa por palavras-chave que funciona deste modo: cada imagem da base de dados está classificada numa categoria, o utilizador escolhe uma destas categorias é efetuada a pesquisa no ficheiro XML e é exibido as 30 primeiras imagens.

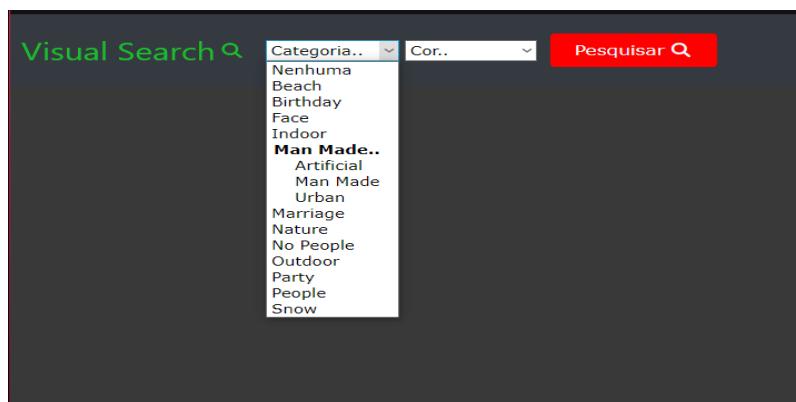


Figura 1: Seleção da categoria

```
//Method to search images based on keywords
searchKeywords(category) {
    const cnv = document.getElementById("canvas");
    //carrega ficheiro xml com a informaçao das img
    let xmlDoc = this.XML_db.loadXMLfile(this.XML_file);
    //guarda os paths das 30 primeiras
    let Images_path = this.XML_db.SearchXML(category, xmlDoc, 30);
    //Exibe
    this.gridView(cnv,Images_path);
    // this method should be completed by the students
}
```

Figura 2: Método "searchKeywords"

Para procurar as imagens na base de dados utilizámos as funções, já presentes no código base (Figura 2), "loadXMLfile" para carregar e guardar numa variável o documento XML e "SearchXML" que nos permite especificar qual a categoria que queremos, o documento a percorrer e a quantidade de imagens que queremos devolvendo os "paths" de cada imagem. Por fim apresentamos as imagens.

2.2 Visualização das Imagens

Para testarmos o método criado no ponto acima decidimos elaborar a visualização de imagens. Esta função vai receber o canvas onde vão ser desenhadas as imagens e um array com os "paths" das imagens a serem desenhadas. Começamos

```
//até 7 pois vai ser o numero de linhas a fazer na grid
for (let i = 0; i <= 7; i++) {
    //horizontais
    const x = i*imgWidth;
    ctx.moveTo(x, 0);
    ctx.lineTo(x, canvas.height);
    ctx.stroke();

    //verticais
    const y = i*imgHeight;
    ctx.moveTo(0, y);
    ctx.lineTo(canvas.width, y);
    ctx.stroke();
}
//5 imagens por linha 6 por coluna
for (let yCell = 0; yCell < 6; yCell++) {
    for (let xCell = 0; xCell < 5; xCell++) {
        const x = xCell * imgWidth;
        const y = yCell * imgHeight;
        //cria a imagem com a posição e dimensão calculada
        let img = new Picture(x+p, y+p, imgWidth-p*2, imgHeight-p*2,Images_path[count], "test");
        //exibe
        img.draw(canvas);
        //incrementa o indice
        count++;
    }
}
```

Figura 3: Método "gridView"

por limpar o canvas, inicializar um contador, que vai servir de índice das imagens a percorrer sendo incrementado quando uma é colocada na grelha, e estabelecer as dimensões das imagens que vamos mostrar, de seguida fazemos a grelha onde as imagens vão ser expostas sendo esta composta por 7 linhas pois vão existir 5 imagens por linha e 6 por coluna. Por fim percorremos o número de colunas e o número de linhas criando a imagem (no índice do contador) na posição correta e desenhando-a na grelha. A ordem das imagens é da esquerda para a direita por linha.

2.3 Lista de cores

Antes de desenvolvêrmos a pesquisa por cor precisávamos de uma forma de o utilizador escolher a cor que deseja. Para resolver este problema damos ao utilizador uma lista de cores com o nome da cor e com um círculo dessa mesma cor, ilustrado na Figura 4.

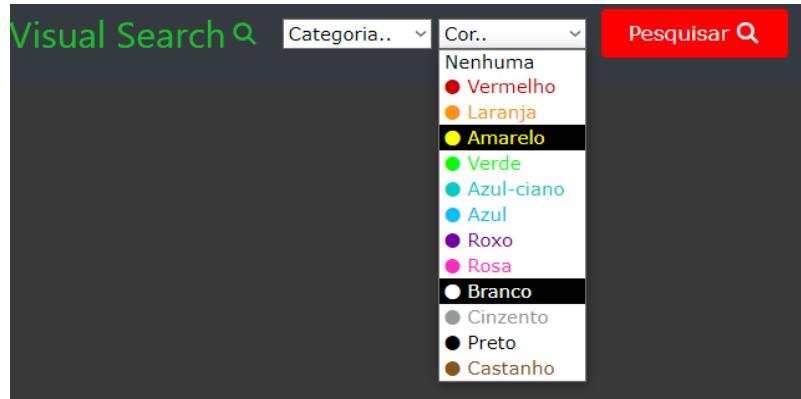


Figura 4: Seleção da cor

Como as cores branco e amarelo eram pouco visíveis num fundo branco decidimos mudar o seu fundo para preto. A cor do círculo e a cor da fonte são iguais as cores pretendidas pelo docente. Quando seleccionávamos uma cor esta opção tinha o mesmo visual que tinha na lista, como solução adicionámos a linha de código destacada na Figura 5. Esta linha faz com que a opção selecionada, quando a lista está fechada, tenha a mesma cor da opção que o utilizador escolheu ao mudando a sua cor para a correspondente através do valor da opção e de css criadas para cada uma das cores.

```
<select class="select" name="color_pick" id="sel" style="height: 25px;"  
       onclick="document.getElementById('sel').className = 'text-' + document.forms['pesquisa']['color_pick'].value;">  
    <option value="none" hidden selected disabled> Cor..</option>  
    <option value="none"><span style="color: □#000;"> Nenhum(a)</span></option>  
    <option value="red" class="text-red"> &#11044; Vermelho</option>  
    <option value="orange" style="color: □rgb(251,148,11);"> &#11044; Laranja</option>  
    <option value="yellow" style="color: □rgb(255,255,0); background-color: □#000;"> &#11044; Amarelo</option>  
    <option value="green" style="color: □rgb(0,255,0);"> &#11044; Verde</option>  
    <option value="Blue-green" style="color: □rgb(0, 204, 198);"> &#11044; Azul-ciano</option>  
    <option value="blue" style="color: □rgb(3,192,255);"> &#11044; Azul</option>  
    <option value="purple" style="color: □rgb(118,0,167);"> &#11044; Roxo</option>  
    <option value="pink" style="color: □rgb(255,44,191);"> &#11044; Rosa</option>  
    <option value="white" style="color: □rgb(255,255,255); background-color: □#000;"> &#11044; Branco</option>  
    <option value="grey" style="color: □rgb(153,153,153);"> &#11044; Cinzento</option>  
    <option value="black" style="color: □rgb(0,0,0);"> &#11044; Preto</option>  
    <option value="brown" style="color: □rgb(136,84,24);"> &#11044; Castanho</option>  
</select>
```

Figura 5: Lista de Cores - HTML

2.4 Pesquisa por Cor

De longe a "feature" mais difícil de desenvolver foi a pesquisa por cor, muito por culpa da nossa ambição de querer fazer através do histograma de cores e não pela cor dominante. Para obter o histograma precisamos de percorrer cada pixel de cada imagem e determinar a qual das 12 cores pertence. Cada imagem vai ser representada por 12 números que representa o número de ocorrências de cada uma das cores na imagem.

```
//percorre os pixels duma imagem
for (let i = 0, n = pixels.data.length; i < n; i+=4) { // 4 = h,s,v,alpha (por isso +=4)

    //percorre o num de cores
    for (let j = 0; j < 12; j++){

        //modulo da diferença do rgb predefinido e rgb do pixel
        let difr = Math.abs(pixels.data[i+0] - this.redColor[j]);
        let difg = Math.abs(pixels.data[i+1] - this.greenColor[j]);
        let difb = Math.abs(pixels.data[i+2] - this.blueColor[j]);
        let diftotal = difr + difg + difb;//soma das diferenças
```

Figura 6: Cálculo do histograma de cores

Para cada pixel da imagem temos que percorrer cada uma das cores, calcular o modulo da diferença do rgb predefinido pelo docente com o rgb desse pixel e por fim fazer a soma de cada uma destas diferenças, como ilustrado na Figura 6. Se estes valores forem inferiores a um limite, que é diferente para cada cor, incrementamos no array dessa cor. Os valores de limite têm que ser diferentes para cada cor pois existia uma discrepância enorme entre o número de imagens consideradas brancas, pretas, castanhas e cinzentas o tornava os resultados pouco interessantes. Para resolvemos isto fomos mais rígidos nestas cores enquanto que nas cores primarias fomos mais liberais. Achamos um meio termo para as cores entre essas. Também adicionámos um "break;" caso estas condições sejam satisfeitas pois sem este(como está no powerpoint do docente) existia a possibilidade do pixel pertencer a mais do que uma cor, o que não é o que queremos pois muitas dessas vezes inflacionava os valores das cores destacadas anteriormente.

Na abertura da página ocorre sempre a computação de cada imagem, neste caso 100 de cada categoria para não tornar a espera muito demorada. Começamos por carregar a informação de cada imagem através da base de dados XML e percorrer cada uma das categorias e cada uma das imagens. Depois é criada cada imagem, através do "path", criado um evento para o processamento de cada imagem e ocorre a computação da imagem, ou seja, é descoberto o histograma. Este trecho de código foi disponibilizado pelo docente e está demonstrado na imagem abaixo (Figura 7).

```
//Percorrer cada categoria para o numero de imagens especificado e fazer a computação de cada imagem
let load = this.XML_db.loadXMLfile(this.XML_file);
for (let i = 0; i < this.categories.length; i++) {
  let x = load.getElementsByClassName(this.categories[i]);
  for (let a = 0; a < this.num_Images; a++) {
    //Percorrer a informação de cada imagem
    for (let b = 0; b < x[a].childNodes.length; b++) {
      let imagePath = x[a].childNodes[b].textContent;
      //se o nome for == a path prosseguir
      if (x[a].childNodes[b].nodeName == "path") {
        let img = new Picture(0, 0, 50, 50,imagePath, "test");
        let eventname = "processed_picture_" + img.impath;
        let eventP = new Event(eventname);
        let self = this;

        document.addEventListener(eventname, function(){
          self.imageProcessed(img);
       },false);
        img.computation(cnvs, h12color, colmoments, eventP);
      }
    }
  }
}
```

Figura 7: Código da computação das imagens

Depois de cada imagem ser processada o evento é ativo e o método da Figura 8 é chamado para verificar se todas as imagens estão processadas. Quando este é o caso é criada uma base de dados, no localStorage, organizada por categoria. Dentro de cada categoria estão organizadas por cor do maior número de pixels para o menor de cada uma das 12 cores.

```
//When the event "processed_picture_" is enabled this method is called to check if all the images are
//already processed. When all the images are processed, a database organized in XML is saved in the localStorage
//to answer the queries related to Color and Image Example
imageProcessed (img) {
  //inserir imagem no array com todas as imagens
  this.allpictures.insert(img);
  //testes
  /*console.log("image processed " + this.allpictures.stuff.length + eventname);
  console.log(img.hist);
  console.log(img.impath);*/
  if (this.allpictures.stuff.length === (this.num_Images * this.categories.length)) {
    this.createXMLColordatabaseLS();
    //this.createXMLIExampledatabaseLS();
  }
}
```

Figura 8: Método "imageProcessed"

Após todas as imagens estarem processadas é chamado o método "createXMLColodatabadeLS" que vai criar a base de dados, em XML, no localStorage. Este vai percorrer o número de imagens guardadas de cada categoria (100) e ordená-los de acordo com a cor e qual tem o maior número de pixels dessa cor.

```
//Percorrer cada categoria para o numero de imagens especificado e ordenar de a
for (let i = 0; i < this.categories.length; i++) {
    this.emptyColors();
    let xml = "<images>\n";
    let x = load.getElementsByClassName(this.categories[i]);
    for (let a = 0; a < this.num_Images; a++) {
        //Percorrer a informação de cada imagem
        for (let b = 0; b < x[a].childNodes.length; b++) {
            //se o nome for == a path prosseguir
            if (x[a].childNodes[b].nodeName == "path") {
                let imagePath = x[a].childNodes[b].textContent;
                //Percorrer todas as imagens e se ambos os paths coincidirem
                //(Pois ocorria o erro das imagens não estarem a ser guardadas
                //inserir no array da sua cor
                for (let l = 0; l < this.allpictures.stuff.length; l++) {
                    if (imagePath == this.allpictures.stuff[l].impath) {
                        this.insertInColorArray(l);
                    }
                }
            }
        }
    }
}
```

Figura 9: Método "createXMLColodatabadeLS"1/2

Temos que percorrer a lista de imagens processadas e a lista de imagens na base de dados e se estas coincidirem inserimos a imagem no array da cor correspondente (Figura 9). Fizemos desta forma pois ocorria o erro das imagens não estarem a ser guardadas pela ordem em que são lidas logo ficavam com informações erradas. Descobrimos qual das cores tem o maior numero de pixels na imagem ao aceder o histograma de cores da imagem e utilizamos "Math.max()" que nos dá essa cor, ilustrado na Figura 10. Depois inserimos no array da cor correspondente e a seguir num array que contém os arrays de todas as cores.

```
//insere a imagem com o indice recebido no array da sua cor dominante
insertInColorArray(l)
{
    //Encontrar o indice com o maior num de pixels
    let max = Math.max(...this.allpictures.stuff[l].hist);
    //Encontrar a cor com o maior num de pixels
    let index = this.allpictures.stuff[l].hist.indexOf(max);

    //console.log(l+" color "+colors[index] );
}
```

Figura 10: Método "insertInColorArray"

Por fim depois de fazermos isto para todas as imagens percorremos o número de cores, ordenamos cada array de cores e ordenamos as imagens de acordo com qual tem o maior numero de pixels dessa cor (Figura11), utilizamos o método "sortByColor"disponibilizado pelo docente. Guardamos as imagens. As imagens ficam organizadas, dentro da categoria, por cor e a única informação que tem é a do "path".

```
//Depois percorrer o número de cores ordenar de acordo com o maior
for (let y = 0; y < allColorArrays.length; y++) {
    this.sortByColor(y,allColorArrays[y]);
    for (let z = 0; z < allColorArrays[y].length; z++) {
        xml += '\t<image class="' + colors[y] + '">';
        xml += "<path>" + allColorArrays[y][z].impath + '</path>';
        xml += "</image>\n";
    }
}
xml += "</images>";
this.LS_db.saveLS_XML(this.categories[i], xml);
```

Figura 11: Método "createXMLColodatabadeLS"2/2

Agora estamos prontos para o utilizador poder realizar a sua procura. Se uma categoria tiver sido selecionada utilizámos o método criado pelo docente, "readLS_XML"para guardar numa variável os dados dessa categoria. De seguida selecionamos os elementos que tem a classe com o nome escolhido pelo utilizador, percorremos o número de ocorrências e guardamos o "path"dessas imagens. Por fim exibimos como na pesquisa por palavras-chave, ou seja, chamamos o método gridView e damos o array com estes "paths".

```
searchColor(category, color) {
    const cnv = document.getElementById("canvas");
    let Images_path = [];
    let arrayColor = [];
    let paths = [];
    //se existir uma categoria selecionada
    if (category != 'none') {
        let x = this.LS_db.readLS_XML(category);
        let xx = x.getElementsByClassName(color);
        //console.log(xx[0].textContent);
        //Percorre a quantidade de imagens com a cor selecionada n
        for (let i = 0; i < xx.length; i++) {
            //guarda o path
            Images_path[Images_path.length] = xx[i].textContent;
            //console.log(Images_path);
        }
    }
}
```

Figura 12: Método "searchColor"

3 Descrição da Aplicação Final

A aplicação final permite ao utilizador visualizar uma seleção de imagens, baseada nas suas escolhas. Este pode escolher uma de 14 categorias e uma de 12 cores. A seleção de cores está numa lista, estilo "Google Images" onde pudemos ver essa mesma cor. A seleção por cor tem em conta o histograma de cores de cada imagem e não a cor dominante disponibilizada na base de dados. As imagens são apresentadas em grelha. Também foi adicionado embora não tenha sido requisitado uma forma de pesquisar as imagens apenas por cor organizado de forma a mostra as com maior número de pixels dessa cor e também podemos guardar uma imagem de qualquer seleção de imagem.

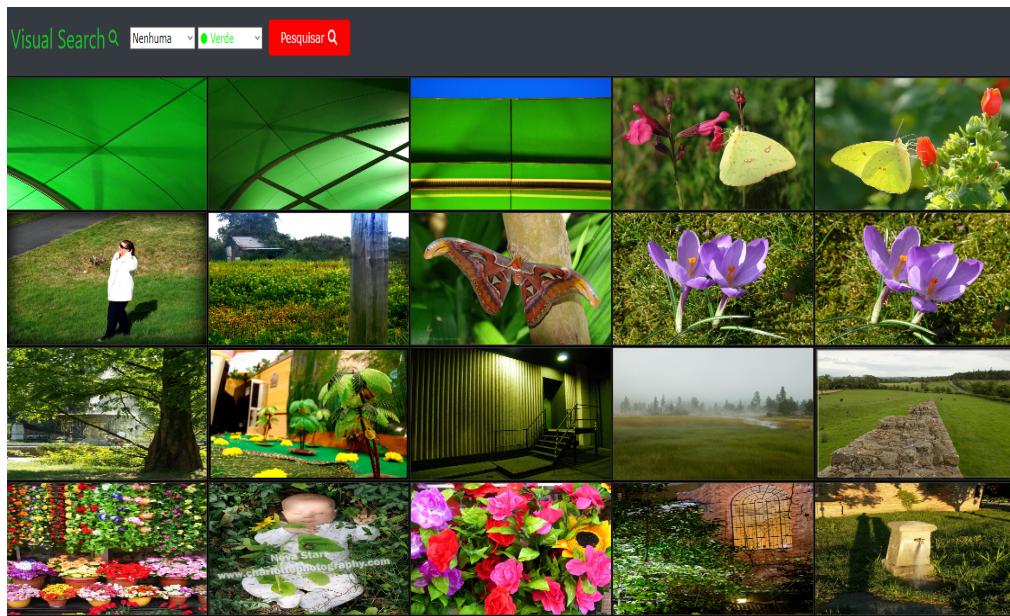


Figura 13: Aplicação Final

4 Discussão das Questões mais Relevantes

Como referido anteriormente, não foi pedido mas adicionámos a possibilidade da procura ser apenas por cor pois achámos que seria uma função que o utilizador pudesse desejar. Para fazermos isto precisamos de ler cada uma das categorias do localStorage e guardar todas os paths de imagens que tenham a mesma cor que o user pede. Depois percorremos esses paths e todas as imagens e quando encontrarmos a imagem coincidente quadramos-a num array. Por fim percorremos esse array, ordenamos e escolhemos as 30 primeiras. Este método encontra-se ilustrado na figura a baixo (Figura 14).

```

//Percorre o num de categorias e faz o mesmo processo que em cima
for (let i = 0; i < this.categories.length; i++) {
    let x = this.LS_db.readLS_XML(this.categories[i]);
    let xx = x.getElementsByTagName(color);
    for (let j = 0; j < xx.length; j++) {
        paths[paths.length] = xx[j].textContent;
    }
}
//Percorre o numero de paths guardados
for (let k = 0; k < paths.length; k++) {
    //Percorre todas as imagens
    for (let l = 0; l < this.allpictures.stuff.length; l++) {
        //Quando encontrar a imagem com o mesmo path guarda essa imagem
        if (paths[k] == this.allpictures.stuff[l].impath) {
            arrayColor[arrayColor.length] = this.allpictures.stuff[l];
            //console.log(this.allpictures[l]);
        }
    }
}
//ordena as imagens por maior num de pixels da cor para o menor
this.sortbyColor(colors.indexOf(color),arrayColor);
console.log(arrayColor);
//guarda as 30 com maior num
for (let m = 0; m < 30; m++) {
    Images_path[Images_path.length] = arrayColor[m].impath;
}

```

Figura 14: Código para escolha de cor sem categoria

Outra melhoria que sugerímos era uma forma de combinar ambas as procura por cor, ou seja, pela cor dominante e pelo histograma. Pois em muitos dos casos as imagens são categorizadas numa categoria de cor em que esta é de facto a cor dominante mas não a cor que salta mais à vista, exemplo Figura 15 em que de facto verde é a cor dominante mas a que se destaca mais com maior consistência de tonalidade é a rosa. Pensamos que um sistema de pesos seja a melhor solução para este problema, em que em vez de todos os pixels valerem o mesmo os que estiverem mais próximos da cor rgb contassem mais.



Figura 15: Exemplo-1

5 Conclusões

O objetivo deste projeto era o de consolidar e aprofundar os conhecimentos adquiridos ao longo de Produção de Conteúdos Multimédia ao conceber-mos uma versão simplificada do "Google Images". Gostávamos de ter tido tempo para adicionar áudio e animações para tornarem a interface mais divertida e intuitiva para o utilizador pois pareceu-nos uma excelente forma de entrelaçar as aprendizagens da parte inicial do semestre nesta disciplina. Porém cremos que as aplicações desenvolvidas e as suas partes facultativas estão de acordo com o pretendido pelos docentes, demonstrando o domínio que temos sobre a matéria lecionada.