



LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA

2022/2023

PRODUÇÃO DE CONTEÚDOS MULTIMÉDIA

Projeto Final - Google images 08/01/2023

Docente: Engº Rui Jesus

Turma: LEIM33D

Realizado por:
Roman Ishchuk A43498
Eduardo Marques A45977

Conteúdo

1	Introdução	2
2	Desenvolvimento da aplicação	3
2.1	Protótipo da aplicação	3
2.2	Desenvolvimento do código	4
2.2.1	Pesquisa por <i>Key word</i>	4
2.2.2	Pesquisa por cor	5
3	Descrição da aplicação	7
4	Discussão de questões relevantes	8
5	Conclusões	10

1 Introdução

Este documento é um relatório para o projeto final da unidade curricular Produção de Conteúdos Multimédia que consiste em explicar detalhadamente o conteúdo, desenvolvimento e finalidade do projeto.

O projeto consiste na criação e implementação um análogo ao *Google Images* com uma base de dados com 1400 imagens com uma ferramenta de *query* em *XML*. As imagens são guardadas localmente, e acessadas através da *metadata* no ficheiro *XML*. A pesquisa de imagens é efetuada através de um sistema *CBIR* (*Content-based Image Retrieval*), qual trabalha em *offline*, e uma *search engine* que avalia a *query* e vai comunicar com a *database*.

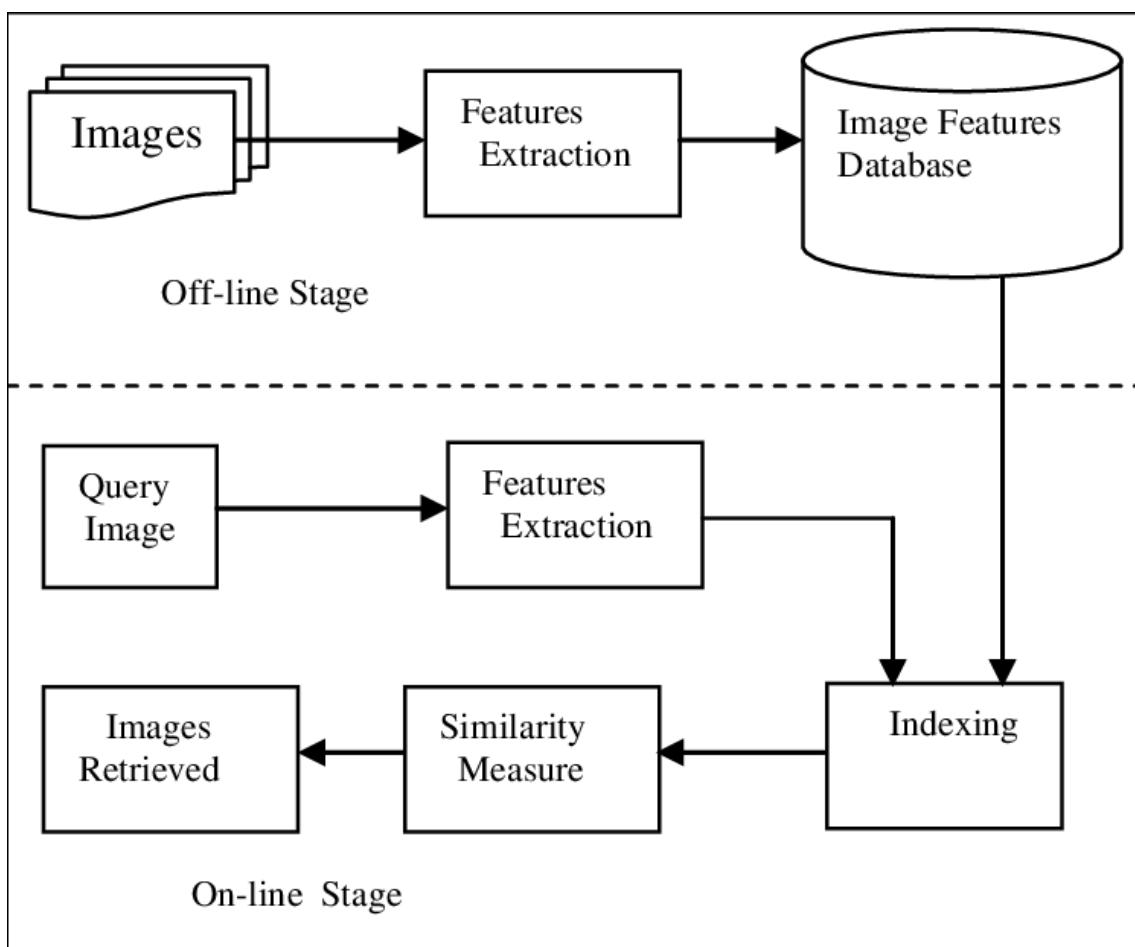


Figura 1: *CBIR*

2 Desenvolvimento da aplicação

2.1 Protótipo da aplicação

Para o desenvolvimento da nossa aplicação, foi desenhado um protótipo para facilitar depois o desenvolvimento da aplicação em si.

Começamos por pensar num título, um título que o utilizador percebesse do que é que a aplicação se trata, mas sem utilizar o nome exclusivo da *Google*. Assim, o grupo chegou ao nome "*Gogol Fortes*", um nome foneticamente parecido a *Google Photos*, mas diferente o suficiente.

Com o título definido, iremos tratar agora da pesquisa da nossa aplicação. Neste campo, iremos ter duas formas de pesquisa, uma por *key word* e uma por cor. Para a organização, chegamos à conclusão que ficaria visualmente mais elegante se a pesquisa por palavra se localizasse a cima da pesquisa por cor.

Por último, teremos que apresentar as imagens ao utilizador. Para tal iremos utilizar um *canvas* onde serão apresentadas um total de 30 imagens, organizadas em 5 colunas e 6 linhas.

Assim, a imagem seguinte é referente ao protótipo da aplicação que irá ser desenvolvida.

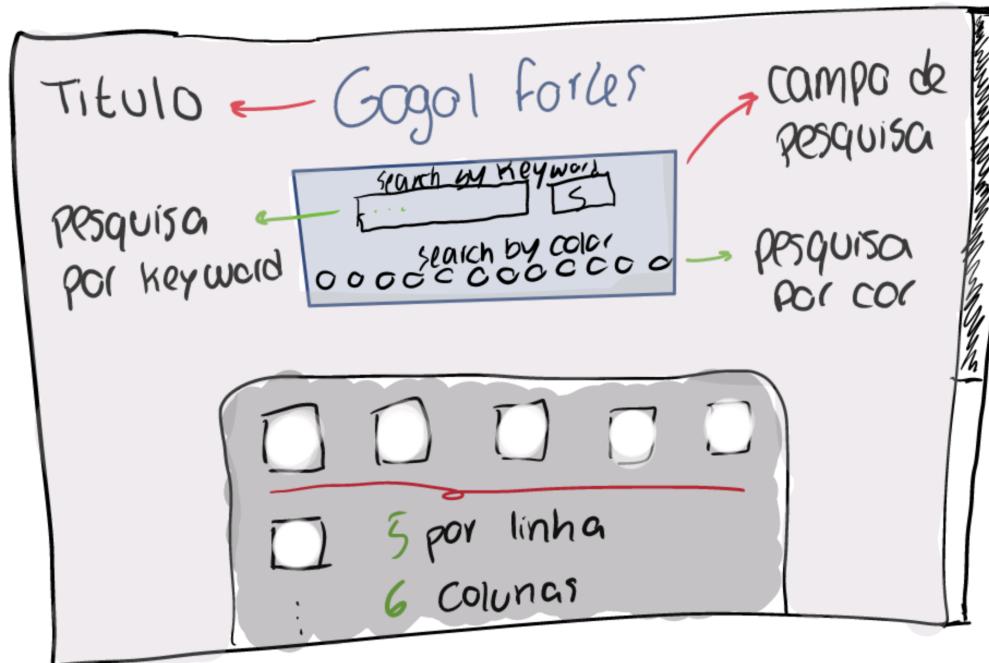


Figura 2: Protótipo da aplicação *Gogol Fortes*

2.2 Desenvolvimento do código

2.2.1 Pesquisa por *Key word*

Para a pesquisa por *Key word*, o código desenvolvido pelo grupo vai procurar ao ficheiro *XML* com a informação referente a cada imagem, fornecido pelos engenheiros da cadeira, e irá apresentar as primeiras 30 imagens da categoria pesquisada pelo utilizador.

São apresentadas 30 imagens apenas pois foi o número que o grupo decidiu usar, porém este número pode ser maior ou menor, visto que a nossas "Imagens relevantes"(método *relevantPictures* do trabalho) cria 100 imagens ao todo com as *paths* fornecidas. São mostradas apenas 30 imagens graças ao método *gridView* que organiza as imagens no *canvas* sempre com a mesma distância entre imagens e com as imagens sempre do mesmo tamanho.

A ordem das imagens é a ordem do número de cada imagem, ou seja, nesta pesquisa as imagens que irão aparecer são da 1 à 30 de cada categoria.

```
searchKeywords(category, canvas) {
  let search_query = this.XML_db.SearchXML(
    category,
    this.XML_doc,
    this.num_Images
  );
  search_query.forEach((element) => {
    this.img_paths.insert(element);
  });
  this.relevantPictures();
  this.gridView(canvas);
}
```

Figura 3: Método *searchKeywords*

```
relevantPictures() {
  const self = this;
  this.allpictures.empty_Pool();
  for (let i = 0; i < this.img_paths.stuff.length; i++) {
    let img = new Picture(
      0,
      0,
      self.imgWidth,
      self.imgHeight,
      self.img_paths.stuff[i],
      "create"
    );
    console.log(this.img_paths.stuff.length);
    self.allpictures.insert(img);
  }
  this.img_paths.empty_Pool();
}
```

Figura 4: Método *relevantPictures*

```
gridView(canvas) {
  let posX = 75;
  let posY = 30;
  let col = 0;
  let maxCol = 5;

  for (let i = 0; i < this.numshownpic; i++) {
    this.allpictures.stuff[i].setPosition(posX, posY);
    this.allpictures.stuff[i].draw(canvas);
    col++;
    posX += 215;
    if (col === maxCol) {
      posY += 175;
      posX = 75;
      col = 0;
    }
  }
}
```

Figura 5: Método *gridView*

2.2.2 Pesquisa por cor

Para a pesquisa de cor ser efetuada, necessitamos que as imagens sejam processadas uma vez no inicio do programa para ser criado uma *Local Storage* com as cores possíveis de pesquisar para cada *Key word*. Nesta *Local Storage*, iram ser atribuídas a cada cor 30 imagens, que podem ser repetidas entre cores diferentes, assim não temos o possível problema de uma cor ter poucas ou nenhuma imagens.

Cada imagem é processada, calculando a diferença de cada pixel com todas as 12 cores. Para isso definimos os limites $limiar1 = 160$ e $limiar2 = 70$. Estes valores pareceram ser os mais adequados, gerando valores que nos são favoráveis. Cada imagem terá um *array hist*, no qual indica o histograma de pixeis mais próximos de cada cor. Após os cálculos, são criados 12 *xml* no *Local Storage*, guardando na *key* a categoria e no *value* o *xml*, onde para cada cor, são apresentados os *paths* de 30 imagens mais próximas.

Calcular histograma de 12 cores de cada imagem



Para cada pixel de uma imagem, calcular a distância de Manhattan entre a cor do pixel e as 12 cores

$$D_1(x, p) = \sum_{i=1}^{12} |x_i - p_i|$$

$$p = \begin{bmatrix} r \\ g \\ b \end{bmatrix} \quad x_{color} = \begin{bmatrix} r_{color} \\ g_{color} \\ b_{color} \end{bmatrix}$$

Se $D_1(x, p) < limiar1 \quad \&\quad |r - r_{color}| < limiar2 \quad \&\quad |g - g_{color}| < limiar2 \quad \&\quad |b - b_{color}| < limiar2$

Figura 6: Cálculo das diferenças

```
count_Pixels(pixels) {
    let allColors = this.redColor.length;
    let allPixels = pixels.data.length;
    let histo = Array(12).fill(0);

    for (let i = 0; i < allPixels; i += 4) {
        let pxr = pixels.data[i];
        let pxy = pixels.data[i + 1];
        let pxb = pixels.data[i + 2];

        for (let j = 0; j < allColors; j++) {
            let difr = Math.abs(this.redColor[j] - pxr);
            let difg = Math.abs(this.greenColor[j] - pxy);
            let difb = Math.abs(this.blueColor[j] - pxb);
            let diftotal = difr + difg + difb;

            if (
                diftotal < this.limiar1 &&
                difr < this.limiar2 &&
                difg < this.limiar2 &&
                difb < this.limiar2
            ) {
                histo[j]++;
            }
        }
    }

    for (let u = 0; u < histo.length; u++) {
        histo[u] = histo[u] / allPixels; //normalizar
    }
    return histo;
}
```

```
createXMLColorDatabaseLS() {
    const self = this;
    console.log(this.allpictures.stuff);
    this.categories.forEach(function (element) {
        //procurar todas as img da categoria
        let xmlRowString = "<images>";
        let imgCategory = self.allpictures.stuff.filter(function (img) {
            return img.category === element;
        });
        self.colors.forEach(function (color) {
            //para cada cor fazer sort
            self.sortbyColor(self.colors.indexOf(color), imgCategory);
            for (let i = 0; i < self.numshownpic; i++) {
                xmlRowString += "<image class=''" + color + "'><path>" +
                    imgCategory[i].impath +
                    "</path></image>";
            }
        });
        xmlRowString += "</images>";
        self.LS_db.saveLS_XML(element, xmlRowString);
        //cria 12 xml no localstorage
    });
}
```

Figura 7: Método *count_Pixels*

Figura 8: Método *createXMLColorDatabaseLS*

Com a *Local Storage* criada, o procedimento para apresentar as imagens ao utilizador no *canvas* é muito parecido à pesquisa apenas por *key word*. Neste caso teremos que utilizar a *key word* e o nome da cor. Com os dois parâmetros temos na nossa posse o necessário para fazer a pesquisa. Começamos por ir à *local storage* e procurarmos a categoria (*key word*) que esta a ser pesquisada pelo utilizador, ao encontrarmos a categoria, pesquisamos no *XML* criado as imagens com o nome da cor na sua classe e inserirmos na variável que irá "guardar" as imagens para depois serem apresentadas.

O resto do procedimento é o mesmo utilizado no ponto anterior com as classes *relevantPictures* e *gridView*, contudo, a classe *relevantPictures* só irá criar 30 imagens em vez de 100 pois o *XML* só é criado com 30 imagens para cada cor.

```
searchColor(category, color, canvas) {
    let xmlDoc = this.LS_db.readLS_XML(category);
    let image_list = this.XML_db.SearchXML(
        color.toLowerCase(),
        xmlDoc,
        this.num_Images
    );
    image_list.forEach((element) => {
        this.img_paths.insert(element);
    });
    this.relevantPictures();
    this.gridView(canvas);
}
```

Figura 9: Método *searchColor*

3 Descrição da aplicação

A aplicação foi desenvolvida de maneira mais *user friendly* possível. Consiste numa barra de pesquisa por *keywords* e uma barra de cores, para pesquisa da categoria com a cor desejada. Os resultados obtidos são demonstrados abaixo da lugar da pesquisa.



Figura 10: *Gogol Fortes*

No total são apresentadas 30 resultados por cada *pesquisa*, e são apresentados numa *grid* (5x6). A pesquisa por cores é efetuada através da cor dominante disponibilizada no *XML*.

Requisitos implementados:

- pesquisa por palavras-chave e por cor (utilizando a cor dominante fornecida no ficheiro XML, 1 valor).
- na pesquisa por cor, esta deverá ser selecionada de uma lista de cores (ver *Google Images*, 1 valor);
- visualização das imagens (resultados das pesquisas) no *canvas*, organizadas em modo *grid* (grelha de imagens) de acordo com a sua relevância para a *query* (1 valor);
- Relatório (1 valor).
- pesquisa por cor utilizando o histograma de 12 cores (esta funcionalidade substitui a pesquisa por cor utilizando a cor dominante fornecida no ficheiro XML, 2 valores);

4 Discussão de questões relevantes

Durante o desenvolvimento deste Projeto final, o grupo deparou-se com algumas dificuldades. A primeira dificuldade foi entender o código fornecido pelos engenheiros da cadeira.

Mesmo sendo uma grande ajuda, quando é dado um código que não é feito por nós é sempre mais complicado para perceber o que está a ser feito e a sua lógica. Contudo, o grupo, com alguma pesquisa e tentativas de perceber o código, o grupo conseguiu ultrapassar este obstáculo.

Outra dificuldade que o grupo se deparou foi com o método *loadXMLfile* da classe *XMLDatabase* fornecida pelos engenheiros. O problema estava na linha de código que abre o ficheiro. Onde deveria estar escrito *GET* estava escrito *POST* mas com alguma pesquisa e ajuda no engenheiro, conseguimos perceber a origem do problema e o mesmo foi corrigido.

```
loadXMLfile(filename) {
    let xmlhttp = {};
    let xmlDoc = {};

    if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
    }
    xmlhttp.open("GET", filename, false);
    xmlhttp.send();
    xmlDoc = xmlhttp.responseXML;
    return xmlDoc;
}
```

Figura 11: Mudança para *GET*

Por ultimo, o grupo deparou-se com um erro onde a cor agora denominada *cyan* não funcionava.

O nome da cor antes da mudança era *blue-green* e a conclusão a que o grupo chega, que pode não ser a conclusão certa, é que visto que temos uma cor de seu nome *blue* e outra *green*, o hífen a meio da palavra *blue-green* deveria baralhar o programa de algum modo que devia tentar ir buscar uma imagem azul e uma imagem verde ao mesmo tempo.

Ao mudarmos o nome da cor e refazermos a *Local storage*, concluímos que passou a ser possível pesquisar imagens com a cor desejada.

```
<div id="colorsearch">
    <button class="color" id="red" value="Red" style="background: #c00; "></button>
    <button class="color" id="orange" value="Orange" style="background: #fb940b; "></button>
    <button class="color" id="yellow" value="Yellow" style="background: #ff0; "></button>
    <button class="color" id="green" value="Green" style="background: #0c0; "></button>
    <button class="color" id="cyan" value="Cyan" style="background: #03c0c6; "></button>
    <button class="color" id="blue" value="Blue" style="background: #00f; "></button>
    <button class="color" id="purple" value="Purple" style="background: #762ca7; "></button>
    <button class="color" id="pink" value="Pink" style="background: #ff98bf; "></button>
    <button class="color" id="white" value="White" style="background: #fff; "></button>
    <button class="color" id="grey" value="Grey" style="background: #999; "></button>
    <button class="color" id="black" value="Black" style="background: #000; "></button>
    <button class="color" id="brown" value="Brown" style="background: #855415; "></button>
</div>
```

Figura 12: Nome da Cor que não funcionava

5 Conclusões

Hoje em dia a pesquisa por imagens faz parte da vida de qualquer pessoa. Basta indicar no motor de pesquisa o pedido e, instantaneamente, aparecem os resultados. Este trabalho ajudou-nos a perceber como, internamente, funciona um motor de pesquisa.

Conseguimos concluir que o processamento de cada imagem só pode ser feito em modo *offline*, pois, se o utilizador pesquisasse, e só depois era feito o processamento, não teria os resultados instantaneamente. Na realidade, o que acontece quando um *user* faz uma *query*, é procurado o *index* da onde as imagens estão guardadas na *data base*.

São apresentadas imagens semelhantes. Isto pode ser alcançado de várias maneiras, por exemplo, elas partilham as mesmas *keywords*, ou por cores parecidas. A pesquisa por cores pode ser alcançada de várias alternativas. Uma delas é, quando as imagens são processadas em regime *offline*, são feitos cálculos matemáticos de cada *pixel*, para atribuir uma cor semelhante a esse pixel. Essa informação é guardada e indexada de maneira a acelerar o processo de busca.

Os motores de busca mais sofisticados, apresentam também outras ferramentas de pesquisa, tais como a pesquisa por tamanho de imagens, tipo de imagem, pesquisa por imagens semelhantes, entre outros. Neste trabalho não foi implementada nenhuma destas opções.