

# Servidor WEB com Persistência

Universidade de Aveiro

João Alcatrão   José Jordão   Mónica Pereira  
Pedro Costa



VERSAO FINAL

# **Servidor WEB com Persistência**

Dept. de Eletrónica, Telecomunicações e Informática  
Universidade de Aveiro

João Alcatrão José Jordão Mónica Pereira Pedro Costa  
(76763) jalcatrao@ua.pt  
(103075) josemmjordao@ua.pt  
(102735) monica.alex.pereira12@ua.pt  
(112682) pedromsc37@ua.pt

18 de junho de 2023

## **Resumo**

Neste relatório encontra-se uma demonstração e explanação, em texto e imagem, da aplicação WEB realizada e baseada no Tutorial 2 – Servidor WEB com Persistência, e complementada com funcionalidades de registo/login e manipulação de imagens.

A nossa aplicação é composta por uma interface (que é em si composta por elementos html, css e javascript) que pode ser utilizada através do browser, uma base de dados (com uso de sql) para efeitos de persistência, e o servidor WEB em si (programada em python).

A interface é utilizada para interagir de maneira simples e elegante com o servidor WEB, através de requests /GET e /POST. O servidor, em prol dos pedidos que recebe da interface, pode atualizar a base de dados, de modo a guardar dados que se desejem guardar, mesmo caso o servidor seja desligado.

A aplicação WEB serve sobretudo para carregar, guardar e visualizar imagens e manipulá-las, permitindo também que sejam feitos comentários e “likes” (e “dislikes”) nessas imagens. Para utilizar estas funcionalidades, é preciso que haja um registo, e um posterior login por parte de quem deseja usufruir delas.

Todas as funcionalidades apresentadas funcionam como pretendemos que funcionem, e todos os objetivos que foram propostos para a realização deste projeto foram alcançados, e ao longo do relatório, iremos demonstrar exemplos que o demonstrem, acompanhados de imagens, tanto do resultado, como do código que o produz, bem como de explicações que visam clarificar como foram implementados.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.0.1	Tema . . . . .	1
1.0.2	Motivação . . . . .	1
1.0.3	Estrutura . . . . .	2
<b>2</b>	<b>Metodologia</b>	<b>6</b>
2.1	Base de Dados . . . . .	6
2.1.1	Tabela accounts – o que é e como interagir com ela . . . . .	8
2.1.2	Tabela images – o que é e como interagir com ela . . . . .	13
2.1.3	Tabelas comments e votes – como é que são usadas . . . . .	22
2.1.4	Manipulação de imagens . . . . .	26
<b>3</b>	<b>Resultados</b>	<b>41</b>
<b>4</b>	<b>Análise</b>	<b>54</b>
<b>5</b>	<b>Conclusões</b>	<b>64</b>

# Capítulo 1

## Introdução

### 1.0.1 Tema

O projeto que nos foi proposto fazer tem como objetivo principal o desenvolvimento de uma aplicação WEB que permita gerir imagens, através do carregamento, visualização, manipulação, e até comentário e apreciação (através de gostos/desgostos) das mesmas.

Mais especificamente, a aplicação deve permitir ao utilizador: carregar imagens à sua escolha com identificação do autor, de um nome que a identifique a imagem, e a data e hora de carregamento da mesma; visualizar as imagens existentes, todas ou filtradas por um autor em particular; visualizar uma imagem em particular, com todos os comentários e votos feitos pelos utilizadores; e acrescentar comentários a imagens. Também deve ter um sistema de pontuação de popularidade de uma imagem através de um sistema de votação de gostos/-desgostos (likes/deslikes). A aplicação deve também permitir a manipulação de imagens através de algoritmos implementados para esse efeito. Finalmente, a aplicação, no nosso caso, também faz uso de um sistema de registo/login.

### 1.0.2 Motivação

Nós começámos este projeto ainda antes de ele nos ter sido proposto, visto que ele baseia-se extensamente no Tutorial 2 – Servidor WEB com Persistência; um tutorial que se encontra para fazer (e que fizemos atempadamente) na parte 2 do Guião 6 da disciplina. Este guiaõ é extensivo e complexo – é preciso ter conhecimento em python, html, css, javascript, e sql para o realizar. Este mesmo conhecimento é assim requerido para realizar a aplicação WEB proposta, que implica também a aplicação de matéria dada e conhecimento obtido em vários outros módulos da disciplina; módulos que se fundamentam na aprendizagem e no uso de matérias acerca de, por exemplo, bases de dados relacionais, da Framework Cherrypy, de criptografia, de testes, e de algoritmos de manipulação de imagem.

### 1.0.3 Estrutura

#### Organização estrutural

A nossa solução é composta por 3 componentes inter-dependentes para assegurar o correto desenvolvimento e o bom funcionamento da aplicação proposta. Estes componentes são: a interface (que contém páginas html, css, e scripts javascript, que permitem ao utilizador interagir com as mesmas, que se encarregam de transmitir os dados recebidos para o servidor WEB para que se possa obter os resultados desejados), o servidor WEB em si (que consiste num conjunto de métodos e classes em python, que conseguem receber os dados da interface através da dinâmica entre os requests recebidos da mesma e da Framework cherypy, e que permite fazer pesquisas e alterações na base de dados, conforme seja pedido, e que ainda permite guardar imagens em pastas do sistema de ficheiros, conforme seja necessário [tmp] ou desejado [uploads]), e a base de dados (relacional, em sql, que confere persistência à aplicação em função dos comandos que o servidor executa sobre ela).

Estas 3 componentes encontram-se organizadas no sistema de ficheiros do sistema operativo (juntamente com pastas auxiliares tmp e uploads) da seguinte forma:

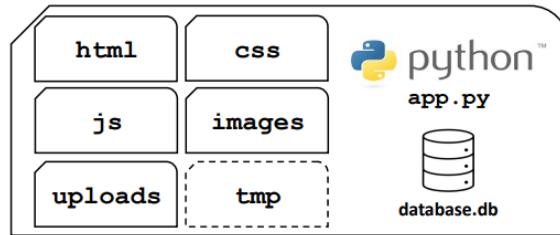


Figura 1.1: Organização dos elementos que constituem a aplicação WEB

## Organização visual

Visto que a interface é o único meio por onde o utilizador pode comunicar com a aplicação por um todo, é importante também definir bem como é que o utilizador pode prosseguir de modo a conseguir obter o que deseja da aplicação.

Tendo em conta o tema e propósito da aplicação de gerir imagens, através do carregamento, visualização, manipulação, e apreciação das mesmas, definimos, na interface, que consiste em páginas html, css, e scripts javascript, e que é acessível através do browser, as seguintes páginas, acessíveis a partir da barra de navegação da página de início da interface: -página de início (Home), cujo propósito é situar o utilizador numa posição confortável, com apenas a informação essencial visível (como o tema da aplicação, e os nomes das páginas que correspondem às funcionalidades de que o utilizador pretenda usufruir);

1. **Gallery Page** - cuja função é listar todas as imagens presentes no sistema, ordenadas por ordem ascendente do nome, permitindo a sua visualização. As imagens em si estarão armazenadas no sistema de ficheiros da aplicação Web. As imagens apresentam o nome dado pelo autor que as carregou, mas no sistema de ficheiros, cada imagem é identificada por um nome criado a partir da síntese do conteúdo da própria imagem. Esta página apresenta assim uma lista de imagens e permite que elas possam ser filtradas pelo nome do autor (total ou parcial).

Quando uma destas imagens é selecionada, é aberta uma nova página autónoma (Image) que apresenta todas as suas propriedades (autor, nome, hora e data de upload no sistema e comentários efetuados pelos utilizadores). Esta página permite que o utilizador acrescente um novo comentário à imagem selecionada, e também apresenta um sistema de apreciação/- pontuação de popularidade da imagem (likes/deslikes).

2. **Upload Page** - página de carregamento de imagens, que permite o carregamento de uma nova imagem no sistema. Cada imagem tem associado um nome e o utilizador que a carregou no sistema, bem como a data e hora de carregamento.

3. **Image Manipulation Page** - página de manipulação de imagens , que permite ao utilizador escolher um método/algoritmo de processamento de imagens, e selecionar a imagem (ou imagens, dependendo do método de manipulação que escolher) que pretende alterar.

É de notar que o utilizador não precisa de carregar as imagens que pretende alterar anteriormente a partir da página de Upload, visto que poderá só querer carregar uma imagem que tenha manipulado a seu gosto, e não carregar a original, e a manipulada (pelo que não tem que escolher uma imagem previamente carregada e guardada no servidor, mas sim uma no seu computador pessoal). No caso de manipulações que requerem múltiplas imagens, também seria pouco agradável ter que carregar todas essas imagens primeiro no servidor antes de as poder manipular, o que poderia causar maior frustração caso o utilizador afinal não gostasse do resultado da manipulação.

Além das imagens, o utilizador pode ainda escolher um conjunto de opções ou inserir algum input para personalizar mais ao seu gosto a imagem manipulada, dependendo do método de manipulação alterada. Manipulação de imagens é algo complexo, com resultados por vezes imprevisíveis e surpreendentes, pelo que nesta página, o utilizador tem muitas mais opções a considerar comparativamente às outras páginas, para que tenha maior controlo sobre o que realmente deseja fazer com as suas imagens.

4. **About Page** - página About, que contém informações sobre nós, os realizadores, da aplicação.

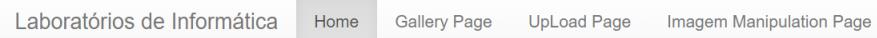


Figura 1.2: Barra de navegação da página de início

É de salientar um ponto importante: esta barra de navegação útil e todas estas funcionalidades de que se pode usufruir a partir da página de início, só são disponibilizadas após haver um login efetuado com sucesso.

Existe assim, uma outra página, que controla o acesso a todas as outras (exceto a About, que tem acesso livre), que é a página de login/registo. A função desta página é permitir aos utilizadores registarem-se e fazerem login, o que os distinguirá dos outros utilizadores. Para não frustrar o utilizador, a efetuação de registo/login é bastante direta, requerendo apenas um nome de utilizador e uma password, e quando um utilizador fizer login e aceder às páginas de Upload, ou clicar numa imagem na Galeria para comentar, verá que o campo de autor já estará preenchido com o seu nome de utilizador – assim, ninguém poderá comentar ou carregar imagens com o nome particular de um utilizador, a não ser o próprio.

# Capítulo 2

## Metodologia

Nesta secção iremos demonstrar como implementámos as diferentes componentes da nossa solução, e explicar como funciona ao certo cada funcionalidade, com exemplos de código para sustentar a explicação.

A estrutura desta secção segue um modelo de ligação entre as diversas componentes que no seu todo constituem uma funcionalidade no seu todo (por exemplo, é indicada uma funcionalidade – digamos, o carregamento de imagens – e explicamos como implementámos esta funcionalidade (começando pela tabela respetiva na base de dados, seguido da criação de elementos html e funções em javascript para obter da interface os dados necessários da imagem em si, e finalmente, no servidor, explicitamos como é que os dados obtidos da interface são manipulados de modo a guardar a imagem no sistema de ficheiros do servidor e a adicionar uma entrada com a informação de nome, autor, data na base de dados, onde se começou).

### 2.1 Base de Dados

Começando pela componente mais simples de implementar e de descrever, a nossa base de dados relacional permite registar diversos dados e informações imprescindíveis ao bom funcionamento da nossa solução, como o registo dos utilizadores e das suas respetivas passwords.

A base de dados é composta por 4 tabelas; uma para guardar informações dos utilizadores (accounts), outra para guardar informações das imagens (images), outra para guardar informações dos comentários (comments), e outra para guardar informações dos likes/dislikes (votes).

A sua implementação mostra-se de seguida.

```
database.sql X
C: > Users > joaoa > OneDrive > Ambiente de Trabalho > Projeto Labi > projeto_com_parametros
1  CREATE DATABASE database;
2  /* Images table */
3  CREATE TABLE images (
4      id INTEGER PRIMARY KEY AUTOINCREMENT, /* primary key */
5      name TEXT,
6      author TEXT,
7      path TEXT,
8      datetime TEXT
9  );
10
11 /* Comments table */
12 CREATE TABLE comments (
13     id INTEGER PRIMARY KEY AUTOINCREMENT, /* primary key */
14     idimg INTEGER, /* foreign key */
15     user TEXT,
16     comment TEXT,
17     datetime TEXT
18 );
19
20 /* Votes table */
21 CREATE TABLE votes (
22     id INTEGER PRIMARY KEY AUTOINCREMENT, /* primary key */
23     idimg INTEGER, /* foreign key */
24     idauthor INTEGER, /*foreign key */
25     ups INTEGER,
26     downs INTEGER
27 );
28
29
30 /* Accounts table */
31 CREATE TABLE accounts (
32     id INTEGER PRIMARY KEY AUTOINCREMENT, /* primary key */
33     username TEXT NOT NULL UNIQUE,
34     password TEXT,
35     user_id TEXT
36 );
```

Figura 2.1: Código de implementação da base de dados.

Analisemos agora com mais detalhe como cada tabela interage com a aplicação, seguindo o modelo descrito acima.

### 2.1.1 Tabela accounts – o que é e como interagir com ela

A tabela accounts contém 4 campos: um id inteiro auto-incrementado, um campo de texto único para os nomes de utilizadores (username), um campo de texto para as passwords (password), e um campo de texto denominado user\_id, que é um parâmetro essencial para garantir que utilizadores diferentes possam aceder à interface em simultâneo e, ao mesmo tempo, garantir a sua individualidade. Este mecanismo de acesso à interface será explicado mais à frente, mas é importante referir desde já que os conteúdos html (além da página do login) não são estáticos. E não o são, porque tal derrotaria o propósito de ter uma página de login/registo, que poderia ser facilmente contornada ao inserir endereços correspondentes a essas páginas por qualquer utilizador, registado e loggado ou não (por exemplo, com a pasta html definida estaticamente no servidor WEB, nas configurações do cherrypy, com o nome “/html”, poder-se-ia muito facilmente aceder à página “gallery.html” inserindo no endereço do browser “127.0.0.1:10013/html/gallery.html”).

Em maior detalhe, na página de login/registo, ao fazer o registo de um utilizador, a interface envia os dados fornecidos pelo utilizador nos campos “Username” e “Password” ao servidor WEB (app.py):

```
<p>Username</p>
<input type="text" name="username" id="username" value="" size="15" maxlength="40" style="color: #blue;"/>
<p>Password</p>
<input type="password" name="password" id="password" value="" size="10" maxlength="40" style="color: #blue;"/>

<br>
<br>
<br>

<p>
    <button onclick="doLogin()" style="background-color: #blue;">Login</button>
    <button onclick="doRegister()" style="background-color: #blue;">Register</button>
</p>

<p><input type="reset" value="Clear" onclick="clearInputs()" style="background-color: #blue;"/></p>
```

Figura 2.2: Organização dos elementos que constituem a aplicação WEB

Ao clicar no botão “Register”, é invocada a função “doRegister()” em javascript, que se encontra em ./js/login.js. A função envia um /POST ao método doRegister do atributo “acions” (classe Actions) do servidor WEB (app.py) com as credenciais como parâmetros:

```
C: > Users > joaoa > OneDrive > Ambiente de Trabalho > Projeto Labi > projeto_com_parametros_nos_enderecos_
1  function doLogin() {
2      sendCredentials("login");
3  }
4  function doRegister() {
5      sendCredentials("register");
6  }
7
8
9  function sendCredentials(login_register) {
10     username = document.getElementById("username").value;
11     password = document.getElementById("password").value;
12
13     if(username == "" || password == "") {
14         alert("Por favor insira um nome de utilizador e uma palavra-passe.");
15     }
16     else {
17         acios_doLogin_doRegister(username, password, login_register)
18     }
19 }
20
21
22 function acios_doLogin_doRegister(username, password, login_register) {
23
24     var data = new FormData();
25     data.append("username", username)
26     data.append("password", password)
27
28     var xhr = new XMLHttpRequest();
29     if (login_register == "login") xhr.open("POST", "/acions/doLogin");
30     else xhr.open("POST", "/acions/doRegister");
31
32     xhr.onreadystatechange = () => {
33         // Call a function when the state changes.
34         if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
35             // Request finished. Do processing here.
36             show_response(JSON.parse( xhr.response ));
37         }
38     };
39
40     xhr.send(data);
41 }
42
43
44 function show_response(response) {
45     if (response.result == "html/inicio.html") {
46         authenticate(response.userID, "inicio");
47     }
48     else {
49         alert(response.result);
50     }
51 }
```

Figura 2.3: Função “doRegister()”. Notar que a função que envia o /POST para registo, é a mesma que o faz para o login, havendo um “if” para as separar

```

657
658     @cherrypy.expose
659     def doRegister(self, username, password):
660
661         db=sql.connect('database.db')
662         result = db.execute("SELECT * FROM accounts WHERE username = ?", (username,))
663         linha = result.fetchone()
664         db.close()
665
666         if linha == None:
667
668             h = hashlib.sha256()
669             h.update(password.encode())
670             password = h.hexdigest()
671
672             s = hashlib.sha256()
673             s.update(password.encode())
674             s.update(username.encode())
675             userID = username+s.hexdigest()
676
677             db = sql.connect('database.db')
678             db.execute("INSERT INTO accounts(username, password, user_id) VALUES (?, ?, ?)", (username, password, userID))
679             db.commit()
680             db.close()
681             return json.dumps({"result" : "Conta criada."}).encode("utf-8")
682
683         else:
684             return json.dumps({"result" : "Erro: a conta já existe."}).encode("utf-8")
685

```

Figura 2.4: Método “doRegister” da classe Actions (atributo .acions da classe Root).

Ao receber as credenciais do utilizador, o método verifica, na tabela “accounts” da base de dados, se este nome de utilizador já existe.

Quer exista quer não exista, o método retorna um objeto json a informar a função javascript que fez o POST que já existe, ou não, uma conta com essa nome; a função simplesmente mostra um alerta na página html a mostrar essa precisa informação. No entanto, se não existir essa conta ainda, o método acede à base de dados e cria essa conta (linhas 677-680 do método “doRegister”).

Note-se ainda que no campo da password, não é inserida a password que o utilizador escreveu, mas sim uma síntese (sha256) da mesma, e que no campo user\_id, é inserido um valor correspondente à concatenação do nome de utilizador (que é único, e como tal, este campo também vai ser único devido a tal) com a síntese da síntese da password.

Este método de segurança é o que permite que cada utilizador navegue segura e unicamente pela interface (após o login) da aplicação, sendo que no endereço, fica escondida a informação da password (visto que é quase impossível “desfazer” uma síntese sha256, quanto mais uma por cima de outra).

Para o caso de um utilizador fazer login, o processo é muito semelhante, mas não há alteração na base de dados, apenas consulta, e se as credenciais forem válidas, o método doLogin retorna um objeto json que a função .js que fez o POST (linhas 45-46), em vez de mostrar um alerta, autentica o utilizador e envia-o para a pagina de inicio.

```

630     class Actions(object):
631         @cherrypy.expose
632         def doLogin(self, username, password):
633
634             h = hashlib.sha256()
635             h.update(password.encode())
636             password = h.hexdigest()
637
638             db=sql.connect('database.db')
639             result = db.execute("SELECT * FROM accounts WHERE username = ?", (username,))
640             linha = result.fetchone()
641             db.close()
642
643
644             if linha == None:
645                 return json.dumps({"result" : "Nome de utilizador ou palavra-passe incorretos."}).encode("utf-8")
646
647             account_info = dict()
648             account_info={"id": linha[0], "username": linha[1], "password":linha[2], "userID": linha[3]}
649
650
651             if account_info["username"].lower() == username.lower() and account_info["password"]==password:
652                 return json.dumps({"result" : "html/inicio.html", "userID": account_info["userID"]}).encode("utf-8")
653
654             return json.dumps({"result" : "Nome de utilizador ou palavra-passe incorretos."}).encode("utf-8")

```

Figura 2.5: Método “doLogin”.

Depois de autenticado, sempre que um utilizador tentar aceder a uma página html, o parâmetro userID, obtido do campo “user\_id” da tabela “accounts”, será usado para verificar a autenticidade do utilizador, e obter o seu “username” no caso de aceder à página Upload ou clicar numa imagem na página Gallery.

```

55     function authenticate(userID, page) {
56         if (page == "inicio") {
57             window.location = 'http://127.0.0.1:10013/page_'+page+'/?userID='+userID;
58         }
59         else {
60             window.open('http://127.0.0.1:10013/page_'+page+'/?userID='+userID, '_blank');
61         }

```

Figura 2.6: Função “authenticate”, responsável por (usando um método do servidor) mudar de página, assegurando a identidade do utilizador que o está a fazer.

A função “authenticate” usa os mecanismos de uso de endereços do cherrypy, mais concretamente, com o cherrypy.expose, para mudar de página. Cada página tem associado um método cujo nome começa por “page\_” e acaba no nome da página em concreto (por exemplo, “page\_gallery” é o método que permite aceder à página da galeria).

Visto que a pasta onde estão as páginas html não está configurada no cherrypy para ser estática, pelas razões de segurança já referidas, usamos métodos para atingir o mesmo objetivo; sendo que estes métodos precisam de um argumento, o “userID” já falado, único a cada utilizador, para validar a mudança de página. Caso o “userID” passado como parâmetro não esteja na base de dados, o método redireciona o utilizador para a página de login/registo.

```

521     #Estas funcionalidades servem para proteger as páginas html de agentes não autenticados
522     @cherrypy.expose
523     def page_inicio(self, userID="None"):
524
525         db=sql.connect('database.db')
526         result = db.execute("SELECT * FROM accounts WHERE user_id = ?", (userID,))
527         linha = result.fetchone()
528         db.close()
529
530         if linha != None:
531             return open("html/inicio.html")
532         else:
533             raise cherrypy.HTTPRedirect("http://127.0.0.1:10013")
534
535
536     @cherrypy.expose
537     def page_gallery(self, userID="None"):
538
539         db=sql.connect('database.db')
540         result = db.execute("SELECT * FROM accounts WHERE user_id = ?", (userID,))
541         linha = result.fetchone()
542         db.close()
543
544         if linha != None:
545             return open("html/gallery.html")
546         else:
547             raise cherrypy.HTTPRedirect("http://127.0.0.1:10013")
548

```

Figura 2.7: Métodos de acesso às páginas, utilizados pela função “authenticate” explicita na figura 2.6.

Visto isto, passemos à tabela “images”, e como é que o servidor interage com ela, e como é que a interface faz com o que o servidor interaja com ela.

### **2.1.2 Tabela images – o que é e como interagir com ela**

A tabela da base de dados que armazena os dados auxiliares de uma imagem na base de dados designa-se images. Os campos desta tabela são: o autor da imagem (author), ou seja o utilizador que carregou a imagem no sistema; o nome que o utilizador lhe atribuiu (name); a data e hora do carregamento da imagem (datetime); e o caminho para a imagem (path), ou seja o diretório onde se encontra o ficheiro. O campo id, como no caso da tabela accounts, representa a chave primária, e é automaticamente incrementado a cada inserção.

Esta tabela é atualizada a partir do carregamento de imagens por parte do utilizador na página “Upload”. Esta página (html) tem um “input” de imagens, cujo clique invoca a função “uploadPhoto”, que mostra a imagem na página, e guarda os dados (binários, em formato File da classe Blob de javascript) da imagem em si numa variável, para posterior envio num POST ao servidor, caso se clique no botão de envio, que chama a função “uploadImage”.

Destaca-se ainda função “autofill”, chamada logo quando a página é carregada, que a partir do endereço da página (que contém o “userID” como parâmetro identificador do utilizador), preenche o campo de nome do autor com o nome do utilizador. Este processo não é direto, sendo efetuado um POST ao método “getUsername” do servidor para “converter” o “userID” no “username” respetivo.

```

<script src="../../js/upload.js"></script>
</head>

<body class="stylador" onload="autofill()">
    <h1>Images Upload</h1>
    <hr>
    <div class="content_image">
        <div class="btn btn-primary btn-block btn-input">
            <input type="file" accept="image/*" onchange="updatePhoto(event)">
            <canvas id="photo" width="550" height="450"></canvas>
        </div>
        <hr>
        <!-- Input buttons for obtain Image Name and Image Author -->
        <div class="btn btn-secondary btn-block btn-input">
            <label for="nameImg">Image Name</label>
            <input type="text" id="nameImg" name="nameImg" required style="color: #blue;">
            &nbsp; &nbsp; &nbsp; &nbsp;
            <label for="authorImg">Image Author</label>
            <input type="text" id="authorImg" name="authorImg" required style="color: #blue;">
            &nbsp; &nbsp;
            <input type="button" class="btn btn-success" value="Upload" onclick="uploadImage()">
        </div>
    </div>
    <div>
        <input type="button" class="btn btn-success" value="Close" onclick="self.close()">
    </div>
</body>

```

Figura 2.8: Página Upload, em html.

```

60 function autofill() {
61     //dar autofocus e readonly ao input do autor, com o nome com que o utilizador fez o login
62     const queryString = window.location.search;
63     const urlParams = new URLSearchParams(queryString);
64     const userID = urlParams.get('userID');
65     var resposta;
66
67     var data = new FormData();
68     data.append("userID", userID)
69
70     var xhr = new XMLHttpRequest();
71     xhr.open("POST", "/getUsername");
72     xhr.send(data);
73
74     xhr.onreadystatechange = () => {
75         if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
76             resposta = (JSON.parse( xhr.response ))
77             const author = resposta.result;
78             document.getElementById("authorImg").value = author;
79             document.getElementById("authorImg").setAttribute("readonly", true);
80         }
81     };

```

Figura 2.9: Função “autofill”, que a partir do parâmetro “userID” do endereço da página, preenche o campo de nome do autor com o nome do utilizador.

```

C: > Users > joaoa > OneDrive > Ambiente de Trabalho > Projeto Labi > projeto_com_parametros_nos_enderecos_se
1 // Image Upload Javascript
2 var file;
3
4 function updatePhoto(event) {
5     var reader = new FileReader();
6     reader.onload = function(event) {
7         //Create an imagem
8         var img = new Image();
9         img.onload = function() {
10             //Put imagen on screen
11             const canvas = $("#photo")[0];
12             const ctx = canvas.getContext("2d");
13             ctx.drawImage(img,0,0,img.width,img.height,0,0,550, 450);
14         }
15         img.src = event.target.result;
16     }
17
18     file = event.target.files[0];
19     //Obtain the file
20     reader.readAsDataURL(file);
21 }
22
23 function uploadImage() {
24     if(file != null) {
25        .sendFile(file);
26         //Release the resources alocated to the selected image
27         window.URL.revokeObjectURL(picURL);
28     }
29     else alert("Imagen em falta!");
30 }
31
32 function sendFile(file) {
33     var data = new FormData();
34     data.append("myFile", file);
35
36
37     //Obtain nameImg and authorImg and fill the form
38     var name = document.getElementById("nameImg").value;           //meter na variável
39     var author = document.getElementById("authorImg").value;        //mesma coisa que
40
41
42     data.append("nameImg", name)                                     //o que é enviado
43     data.append("authorImg", author)                                //agora mete-se
44
45     if (name == "" || author == "") alert("Missing comment and/or username!");
46     else {
47         var xhr = new XMLHttpRequest();
48         xhr.open("POST", "/upload");
49         xhr.upload.addEventListener("progress", updateProgress(this), false);
50         xhr.send(data);
51     }
52 }

```

Figura 2.10: Funções “updatePhoto” (que carrega a imagem na página e guarda os dados numa variável) e “uploadImage”, que envia o POST ao servidor.

Do lado do servidor, temos o método “upload”, responsável por guardar a imagem em si no servidor (com um caminho obtido a partir da junção do caminho da pasta onde se encontra o servidor, mais a pasta “uploads/”, mais o resultado da síntese da imagem) e fazer o registo das suas informações (nome, autor, data, caminho) na base de dados, e o método “getUsername”, que retorna o “username” ao qual o parâmetro de entrada “userID” está associado.

```

46     # Upload image
47     @cherrypy.expose
48     def upload(self, myFile, nameImg, authorImg):      #adicionados parâmetros nameImg, authorImg
49         h = hashlib.sha256()
50
51         filename = baseDir + "/uploads/" + myFile.filename
52         fileout = open(filename, "wb")
53         while True:
54             data = myFile.file.read(8192)
55             if not data: break
56             fileout.write(data)
57             h.update(data)
58         fileout.close()
59
60         ext = myFile.filename.split(".")[-1]
61         # final path of the image and changing the filename
62         path = "uploads/" + h.hexdigest() + "." + ext
63         os.rename(filename, path)
64
65         # nameImg and authorImg are input parameters of this method
66         # obtain the date and time of loading
67         datetime = time.strftime('date:%d-%m-%Y time:%H:%M:%S')
68
69         db = sql.connect('database.db')
70         db.execute("INSERT INTO images(name, author, path, datetime) VALUES (?, ?, ?, ?)", (nameImg, authorImg, path, datetime))
71         db.commit()
72         db.close()

```

Figura 2.11: Método “upload”.

```

602     @cherrypy.expose
603     def getUsername(self, userID):
604
605         db=sql.connect('database.db')
606         result = db.execute("SELECT * FROM accounts WHERE user_id = ?", (userID,))
607         linha = result.fetchone()
608         db.close()
609
610         if linha != None:
611             return json.dumps({"result" : linha[1]}).encode("utf-8")
612         raise cherrypy.HTTPRedirect("http://127.0.0.1:10013")

```

Figura 2.12: Método “getUsername”.

Esta descrição cobre os pontos essenciais da funcionalidade de carregamento de imagens. Passemos agora à funcionalidade de visualização de imagens, na página Gallery, que ainda usa a tabela “images” da base de dados.

A página Gallery mostra todas as imagens carregadas no servidor (em concreto, as que foram carregadas através da página Upload), ordenadas pelo seu nome (nome que o autor deu, e não o de síntese, e que aparece sobre cada imagem, juntamente com o nome do autor), e permite a filtragem das mesmas pelo nome do autor (note-se que pode-se usar sintaxe sql “%autor” para obter matches parciais; note-se ainda que todo o nosso código sql foi feito de maneira a evitar sql injection).

```

20 |     <script src="../js/gallery.js"></script>
21 | </head>
22 |
23 | <body class="stylador">
24 |     <h1>Gallery of Images</h1>
25 |     <br>
26 |     <br>
27 |     <div class="btn btn-secondary btn-block btn-input">
28 |         <label for="authorImg">Author</label>
29 |         <input type="text" id="authorImg" name="authorImg" value="" style="color: #blue;">
30 |         <input type="button" class="btn btn-success" value="Refresh" onclick="imageslist()">
31 |         <p>Nota: pode-se usar % no input para pesquisas que não sejam exatamente o nome do autor, mas que o incluam (ex: %autor)</p>
32 |     </div>
33 |     <br>
34 |
35 |     <div id="showimages"></div>
36 |     <br>
37 |     <br>
38 |
39 |     <div>
40 |         <input type="button" class="btn btn-success" value="Close" onclick="self.close()">
41 |     </div>

```

Figura 2.13: Página Gallery, em html

Note-se que o div com id=“showimages” começa vazio; ele é populado pela função “showimage”, chamada pela “imageslist”, que faz um POST ao método “list” do servidor (com parâmetro de autor), e a partir da resposta deste (que contém os caminhos necessários para representar as imagens na página html através do atributo “src” dos elementos “img”, e que também contém as informações de autor e nome, todas obtidas a partir da tabela “images” da base de dados), itera sobre cada dicionário de informações de imagens recebido, e faz “append” dos elementos (“img” e “h3”) criados nesse div.

```

C:\>User\>java>Gravidez>Ambiente de trabalho>Projeto Labi>projeto_com_parametros_no_endereco_seguro>projeto-final-labi2023g13>project>jp>galleyjs>showimages
1 $(document).ready(
2     function(){
3         imageslist("all");
4     });
5
6     function imageslist(id) {
7         if (id == "all") {
8             if (author == "") author = "all";
9             else {
10                 author = $("#authorImg").val(); //o $ aqui serve como atalho para jquery
11                 if (author == "") author = "all";
12             }
13             $.get("/list",
14                 { id : author },
15                 function(response){
16                     showImages(response);
17                 });
18         }
19
20     function showImages(response) {
21         // response.images is the list of dictionaries with the Images information
22         $('#showimages').html("");
23         for (let i = 0; i < response.images.length; i++) {
24             // HTML code for print the Image information
25             const para = document.createElement("p");
26             const node = document.createTextNode(`Imagem ${response.images[i].name} - Autor: ${response.images[i].author} - Data: ${response.images[i].datetime}`);
27             para.appendChild(node);
28
29             const elexistente = document.getElementById("showimages");
30             elexistente.appendChild(para);
31
32             // HTML code for showing the image and allow to click on it and invoke function showImageComments
33             let img = document.createElement("img"); //esta linha cria uma imagem, que será apresentada como a imagem que tiver o caminho já abaixo especificado
34             img.src = `./${response.images[i].path}`; //Lembrar! caminho da imagem. Como o response.images é uma lista de dicionários, pode-se aceder a um elemento diretamente como se fosse um array
35             img.onclick = function () { showImageComments(response.images[i].id) }; //a propriedade onclick recebe uma função, por isso cria-se uma função nova (sem nome, o javascript permite isso)
36
37             //img.height = 550;
38             //img.width = 450;
39
40             elexistente.appendChild(img);
41
42         }
43
44         //o div na página gallery.html com id=showimages começa inicialmente vazio. Esta função cria novos elementos (parágrafos, headers, imagens) e os append destes a esse div inicial. Estes novos

```

Figura 2.14: Função “imagelist”, que faz um POST ao servidor com a informação de autor a filtrar, e a função “showimages”, que a partir da resposta do servidor (que depende também da base de dados), adiciona as imagens, os seus nomes, e os nomes dos seus autores à página html.

```

79     @cherrypy.expose
80     def list(self, id):
81         db = sql.connect('database.db')
82         if (id == "all"):
83             result = db.execute("SELECT * FROM images")
84         else:
85             result = db.execute("SELECT * FROM images WHERE author LIKE ? ", (id,)) #usar '%' no campo do autor
86         rows = result.fetchall()
87         db.close()
88
89         # Generate result (list of dictionaries) from rows (list of tuples)
90         result = []
91         for linha in rows:
92             dicioImagem={"id": linha[0], "name": linha[1], "author":linha[2], "path":linha[3], "datetime": linha[4]}
93             result.append(dicioImagem)
94
95
96         # eventually sort result by image name before return
97         result.sort(key=lambda x: x["name"].lower())
98
99         cherrypy.response.headers["Content-Type"] = "application/json"
100        return json.dumps({"images": result}).encode("utf-8")

```

Figura 2.15: Método “list” do servidor, que procura na tabela “images” da base de dados as informações das imagens cujo autor corresponde ao parâmetro indicado, e retorna essas informações, ordenadas, numa lista de dicionários em formato json.

Esta descrição tenta demonstrar o essencial sobre a implementação da funcionalidade de visualizar imagens, mas nesta mesma página da Galeria, pode-se aceder a outra funcionalidade: a de ver os comentários e gostos/desgostos de uma imagem em particular. Na Fig16, linha 37, demonstra-se a atribuição de um evento “onclick” a cada imagem. Este evento permite que, ao clicar numa imagem, seja aberta uma nova página, dedicada a esta nova funcionalidade.

```
51  function showimagecomments(id) {
52
53      function get_page() {
54
55          const queryString = window.location.search;
56          const urlParams = new URLSearchParams(queryString);
57          const userID = urlParams.get('userID')
58
59          authenticate(userID)
60      }
61
62      get_page()
63
64      function authenticate(userID) {
65          window.open('http://127.0.0.1:10013/page_image/?id=' + id + '&userID=' + userID, '_blank');
66      }
67 }
```

Figura 2.16: Definição da função “showimagecomments”.

A função “showimagecomments” que é atribuída a cada imagem mostrada na galeria, confere acesso à funcionalidade de ver e fazer comentários e gostos e desgostos à imagem.

Para este efeito, ela recebe um parâmetro de id, correspondente ao id da imagem selecionada na tabela “images” da base de dados (onde está o seu caminho, que é essencial para a sua visualização), e acede ao método “page\_image” do servidor (GET) para obter a página image.html.

```

20 |     <script src="../../js/image.js"></script>
21 | </head>
22 |
23 | <body style="font-size: 15px" class="stylador">
24 |     <h1>Image and Comments</h1>
25 |     <hr>
26 |     <div class="row">
27 |         <div class="columnleft">
28 |             <h3>Image Information</h3>
29 |             <hr>
30 |             <div id="imageinfo"></div>
31 |         </div>
32 |
33 |         <div class="columnright">
34 |             <h3>Users Comments</h3>
35 |             <hr>
36 |             <div id="comments" style="border: 3px solid"></div>
37 |         </div>
38 |     </div>
39 |
40 |     <hr>
41 |     <div class="btn btn-secondary btn-block btn-input">
42 |         <label for="user">User Name</label>
43 |         <input type="text" id="user" name="user" required style="color: #blue;">
44 |         &nbsp; &nbsp;
45 |         <label for="comment">Comment</label>
46 |         <input type="text" id="comment" name="comment" required style="color: #blue;">
47 |         <input type="button" class="btn btn-success" value="Send Comment" onclick="newcomment()">
48 |     </div>
49 |
50 |     <div>
51 |         <span id="thumbs_up" style="font-weight:bold; font-size:20px;"></span>
52 |         
53 |         
54 |         <span id="thumbs_down" style="font-weight:bold; font-size:20px;"></span>
55 |     </div>
56 |
57 |     <hr>
58 |     <div>
59 |         <input type="button" class="btn btn-success" value="Close" onclick="self.close()">
60 |     </div>
61 | </body>

```

Figura 2.17: Página Image, em html.

Embora não seja aqui mostrado na página html, na função javascript respetiva, o id da imagem é obtido logo a partir do parâmetro “id” do endereço assim que a imagem é carregada (obtendo-se assim um parâmetro essencial para obter o caminho, o nome, e o autor da imagem, bem como a data e a hora na qual foi carregada).

```

C:\> Users > joaoa > OneDrive > Ambiente de Trabalho > Projeto Labi > projeto_com_parametros_nos_enderecos_seguro > projeto-final>
1  var id;
2
3  $(document).ready(
4      function(){
5          const params = new URLSearchParams(window.location.search);
6          id = params.get("id");
7
8          //dar autofocus e readonly ao input do autor, com o nome com que o utilizador fez o login
9          const queryString = window.location.search;
10         const urlParams = new URLSearchParams(queryString);
11         const userID = urlParams.get('userID');
12         var resposta;
13
14         var data = new FormData();
15         data.append("userID", userID)
16
17         var xhr = new XMLHttpRequest();
18         xhr.open("POST", "/getUsername");
19         xhr.send(data);
20
21         xhr.onreadystatechange = () => {
22             if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
23                 resposta = (JSON.parse( xhr.response ))
24                 const username = resposta.result;
25                 document.getElementById("user").value = username;
26                 document.getElementById("user").setAttribute("readonly", true);
27
28             }
29         };
30
31
32
33
34         imagecomments ();
35
36     });

```

Figura 2.18: Função javascript imagem.js; função “onload” da página respetiva.

Também é obtido o “username” do utilizador a partir do “userID” no endereço, de maneira semelhante à que já foi descrita, e que é usado para preencher o campo de autor de comentário, que corresponde ao elemento com id=“user” na imagem acima.

### 2.1.3 Tabelas comments e votes – como é que são usadas

As informações sobre os comentários e votos da imagem selecionada são obtidos a partir da tabela “comments” da base de dados, que é feito a partir da função “imagecomments”, que faz um POST ao método “comments” do servidor, cuja resposta é manipulada pela função “showimageandinfo”, de modo a criar os elementos html necessários para mostrar todos os comentários e votos obtidos na página.

```
38   function imagecomments() {
39     $.get("/comments",
40       { idimg : id },
41       function(response){
42         | showimageandinfo(response);
43       }));
44   }
45
46   function showimageandinfo(response) {
47     // response.image is the image information
48     document.getElementById("imageInfo").textContent='';           //limpar div, para atualizar com os novos valores (ao inicio não tem efeito, pois o div não tem elementos, mas quando
49
50     let img = document.createElement("img");
51     img.src = "./"+response.image["path"];
52     img.height = 550;
53     img.width = 900;
54
55     let img_info = document.createElement("h2");
56     let texto_node = document.createTextNode("imagem "+response.image["id"]+" Nome: "+response.image["name"]+" Author: "+response.image["author"]+" "+response.image["datetime"]);
57
58     document.getElementById("imageInfo").appendChild(img_info.appendChild(texto_node));
59     document.getElementById("imageInfo").appendChild(img);
60
61
62     // response.comments is the image list comments
63     document.getElementById("comments").textContent='';           //limpar div, para atualizar com os novos valores (ao inicio não tem efeito, pois o div não tem elementos, mas quando
64
65     for (let i=0; i<response.comments.length; i++) {
66       let user_date = document.createElement("h3");
67       let name_time = document.createTextNode(response.comments[i]["user"]+" "+response.comments[i]["datetime"]);
68       user_date.appendChild(name_time);
69
70       let comment = document.createElement("h4");
71       let texto = document.createTextNode(response.comments[i]["comment"]);
72       comment.appendChild(texto);
73
74       let section = document.createElement("div");
75       document.getElementById("comments").appendChild(section.appendChild(user_date));
76       document.getElementById("comments").appendChild(section.appendChild(comment));
77     }
78
79     // response.votes is the image votes
80     document.getElementById("thumbs_up").textContent="";
81     document.getElementById("thumbs_down").textContent="";
82
83     let ups = document.createElement("p");
84     let ups_number = document.createTextNode(response.votes["ups"]);
85
86     let downs = document.createElement("p");
87     let downs_number = document.createTextNode(response.votes["downs"]);
88
89     document.getElementById("thumbs_up").appendChild(ups.appendChild(ups_number));
90     document.getElementById("thumbs_down").appendChild(downs.appendChild(downs_number));
```

Figura 2.19: Funções “imagecomments” e “showimageinfo”.

Do lado do servidor, o método “comments” procura em 3 tabelas distintas da base de dados (images, votes e comments) as informações necessárias para que se possa visualizar a imagem (com o seu caminho), o seu nome, o seu autor, a hora e data a que foi submetida, bem como as informações de todos os comentários e gostos e desgostos feitos relativamente a esta imagem.

Só há uma imagem a ser representada, e os votos são aglomerados em valores totais (não é mostrado quem meteu gosto ou desgosto, simplesmente se mostra o valor total de cada métrica), pelo que um dicionário para cada é suficiente para representar essa informação. Os comentários podem ser diversos, e importa saber quem os fez, por isso é necessária uma lista de dicionários para representar esta informação.

Toda ela é posteriormente devolvida num objeto json, que é extensivamente usado pela função “showimageandinfo” para meter em html estes dados.

```

104     # List comments
105     @cherrypy.expose
106     def comments(self, idimg):
107         db = sqllite.connect('database.db')
108         # result = db.execute(query of type SELECT for image of the id idimg)
109         result = db.execute("SELECT * FROM images WHERE id=?", (idimg,))
110         linha = result.fetchone()
111
112         # Generate output dictionary with image information
113         imageinfo = dict()
114         imageinfo={"id": linha[0], "name": linha[1], "author":linha[2], "path":linha[3], "datetime": linha[4]}
115
116
117         # result = db.execute(query of type SELECT for all comments of the id idimg)
118         result = db.execute("SELECT * FROM comments WHERE idimg = ?", (idimg,))
119         rows = result.fetchall()
120
121
122         # Generate output dictionary with image comments list
123         comments = []
124         for linha in rows:
125             dicioComment={"id": linha[0], "idimg": linha[1], "user":linha[2], "comment":linha[3], "datetime": linha[4]}
126             comments.append(dicioComment)
127
128         # result = db.execute(query of type SELECT for votes of the id idimg)
129         result = db.execute("SELECT * FROM votes WHERE idimg = ?", (idimg,))
130         linhas = result.fetchall()
131         db.close()
132
133
134         # Generate output dictionary with image votes
135         imagevotes = dict()
136
137         if linhas == []:
138             imagevotes={"ups":0, "downs":0}
139         else:
140             gostos = 0
141             desgostos = 0
142             for linha in linhas:
143                 if linha[3]==1:
144                     gostos+=1
145                 if linha[4]==1:
146                     desgostos+=1
147             imagevotes={"ups":gostos, "downs":desgostos}
148
149
150         cherrypy.response.headers["Content-Type"] = "application/json"
151         return json.dumps({"image": imageinfo, "comments": comments, "votes": imagevotes}).encode("utf-8")

```

Figura 2.20: Método “comments”.

Para terminar esta secção da funcionalidade de apreciação da foto, não basta descrever como é que se pode ver essa informação, mas também como é que se pode adicionar comentários e gostos à imagem selecionada.

A página html tem campos de texto para escrever comentários (o campo de autor é automaticamente preenchido, como já referido), com o devido botão de submeter, e tem também pequenas imagens que representam os likes e dislikes, que ao carregar neles, faz-se gosto e desgosto, respetivamente. Carregar novamente nestes botões, retira o gosto ou desgosto feito, conforme é usual acontecer em aplicações que permitem esta funcionalidade (como o Facebook e o Instagram).

```

94 function newcomment() {
95     // obtain the user and comment from image page
96
97     var user = document.getElementById("user").value;           //escrever aqui o texto que tiver escrito no formulário quando o clique no botão que diz
98     var comment = document.getElementById("comment").value;
99
100    if (user == "" || comment == "") alert("Missing comment and/or username!");
101    else {
102        $.post("/newcomment",
103            { iding: id, username: user, newcomment: comment },      //antes estava aqui escrito, e nas funções upvote() e downvote() abaixo também, idimage
104            function() { imagecomments(); });
105    }
106}
107
108 function upvote() {
109    const queryString = window.location.search;
110    const urlParams = new URLSearchParams(queryString);
111    const userID = urlParams.get('userID');
112
113    $.post("/upvote",
114        { iding: id, userID: userID },
115        function(response)
116        {
117            // update thumbs_up and thumbs_down
118            document.getElementById("thumbs_up").textContent = parseInt(document.getElementById("thumbs_up").textContent)+JSON.parse(response).result;
119        });
120}
121
122 function downvote() {
123    const queryString = window.location.search;
124    const urlParams = new URLSearchParams(queryString);
125    const userID = urlParams.get('userID');
126
127    $.post("/downvote",
128        { iding: id, userID: userID },
129        function(response)
130        {
131            // update thumbs_up and thumbs_down
132            document.getElementById("thumbs_down").textContent = parseInt(document.getElementById("thumbs_down").textContent)+JSON.parse(response).result;
133        });
134}

```

Figura 2.21: Funções “newcomment”, “upvote” e “downvote”.

Sempre que é submetido um novo comentário (é invocada a função “newcomment”, que faz POST ao método com o mesmo nome no servidor), a div que os mostra é apagada de todo o seu conteúdo, pois será feita uma nova chamada à função “imagecomments”, e posteriormente “showimageandinfo”, que já foram descritas, para mostrar todos os comentários.

```

156     @cherrypy.expose
157     def newcomment(self, idimg, username, newcomment):
158         datetime = time.strftime('date:%d-%m-%Y time:%H:%M:%S')
159
160         db=db.connect('database.db')
161         db.execute("INSERT INTO comments(idimg, user, comment, datetime) VALUES (?, ?, ?, ?)", (idimg, username, newcomment, datetime))
162         db.commit()
163         db.close()
164
165
166
167     @cherrypy.expose
168     def upvote(self, idimg, userID):
169         db=db.connect('database.db')
170         authorID = db.execute("SELECT * FROM accounts WHERE user_id = ?", (userID,)).fetchone()[0]
171         result = db.execute("SELECT * FROM votes WHERE idimg = ? and idauthor = ?", (idimg,authorID))
172         linhas = result.fetchone()
173         db.close()
174
175         db=db.connect('database.db')
176         if linhas == None:
177             db.execute("INSERT INTO votes(idimg, idauthor, ups, downs) VALUES (?, ?, ?, ?)", (idimg, authorID, 1, 0))
178             db.commit()
179             db.close()
180             return json.dumps({"result": 1}).encode("utf-8")
181         else:
182             db.execute("DELETE FROM votes WHERE idimg = ? and idauthor = ?", (idimg, authorID))
183             if linhas[3]==1:
184                 db.execute("INSERT INTO votes(idimg, idauthor, ups, downs) VALUES (?, ?, ?, ?)", (idimg, authorID, 0, linhas[4])) #retirar gosto
185                 db.commit()
186                 db.close()
187                 return json.dumps({"result": -1}).encode("utf-8")
188             else:
189                 db.execute("INSERT INTO votes(idimg, idauthor, ups, downs) VALUES (?, ?, ?, ?)", (idimg, authorID, 1, linhas[4])) #meter gosto
190                 db.commit()
191                 return json.dumps({"result": 1}).encode("utf-8")
192

```

Figura 2.22: Métodos “newcomment” e “upvote” (o método “downvote” é muito semelhante ao método "upvote", só que atualiza os desgostos em vez dos gostos).

Para concluir a secção da metodologia, demonstremos agora como implementámos a única funcionalidade que ainda não explicitámos (e a única que não faz uso da base de dados): a manipulação de imagens

## 2.1.4 Manipulação de imagens

A funcionalidade de manipulação de imagens da nossa aplicação é muito complexa, mesmo não usando a base de dados. A base de dados não é usada, porque as imagens que o utilizador pretender manipular, são carregadas a partir do seu computador/dispositivo, e estas já contém os dados necessários para que os nossos métodos de manipulação de imagem funcionem.

O utilizador, ao carregar uma imagem na página Image Manipulation, e ao submeter a imagem para manipulação, faz com que esta, caso seja validada a par com os outros “inputs” pedidos pelo método de manipulação escolhido, faz com que a imagem seja guardada na pasta ./tmp da aplicação, para que os seus dados possam ser usados pelos métodos do servidor (que são definidos num ficheiro python complementar, e importados e usados no programa principal em si) para que a manipulação da imagem ocorra. Caso a manipulação seja bem sucedida, a imagem resultante também é guardada na pasta ./tmp, e é mostrada na página html através de funções javascript.

Caso o utilizador goste da imagem que manipulou, pode simplesmente guardá-la a partir da página ao clicar com o botão direito (ou usando outro método que alcance o mesmo resultado) e escolher “Guardar imagem”. Se posteriormente pretender com que esta imagem manipulada seja mostrada na página da Galeria, pode carregá-la na página Upload, mas não é obrigado a carregar primeiro a imagem original que deseja manipular (ou imagens, caso o método de manipulação o peça) no servidor (embora o possa fazer à vontade).

Ao entrar na página Image Manipulation, encontram-se 2 secções distintas, lado a lado: uma, do lado esquerdo, para escolher uma imagem, e outra, do lado direito, que é usada para mostrar a imagem manipulada.

Mas para manipular a imagem, é preciso escolher um método, a partir dum lista opções sob a forma de elemento “select” que se encontra no centro da página, juntamente com o botão de submeter, mas abaixo das 2 secções mencionadas acima. Ao selecionar um método de manipulação, podem aparecer, conforme o método, outros “inputs” para o utilizador escolher/preencher abaixo do botão de submeter, ou pode até aparecer, do lado esquerdo, abaixo da secção de “input” da imagem original a escolher, outro “input” de imagem para o utilizador usar.

```

C:\Users\joaoa\OneDrive\Ambiente de Trabalho\Projeto Labi\projeto_com_parametros_nos_enderecos_seguro\projeto-final-labi2023g13\project\html> image
21   <script src="../js/image_handler.js" type="module"></script>
22 </head>
23
24 <body style="font-size: 15px" class="stylador">
25   <h1>Image Manipulation</h1>
26   <hr>
27   <div class="row">
28     <div class="columnleft">
29       <h3>Original Image</h3>
30       <br>
31       <div id="original_image">
32         <input type="file" accept="image/*" onchange="handler_updatePhoto(event)">
33         <canvas id="photo" width="550" height="450"></canvas>
34       </div>
35     </div>
36
37     <div class="columnright">
38       <h3>Manipulated Image</h3>
39       <br>
40       <div id="manipulated_image" style="border: 3px solid"></div>
41     </div>
42   </div>
43
44   <div>
45     <form>
46       <label for="algo">Choose an algorithm:</label>
47       <select id="algo" name="Algorithm" style="color: blue;" required onchange="handler_im_modificador(event)">
48         <option value="">--Please select an algorithm--</option>
49         <option value="Watermark">Watermark</option>
50         <option value="Vignette">Vignette</option>
51         <option value="Bordador">Borderer</option>
52         <option value="Sepsis/Lomography">Sepsis/Lomography</option>
53         <option value="Saturation">Saturation</option>
54         <option value="Gamma">Gamma</option>
55         <option value="Intensifier">Intensifier</option>
56         <option value="Resizer">Resizer</option>
57         <option value="Unknown">Unknown</option>
58       </select>
59       <br><br><br>
60       <input type="button" class="btn btn-success" value="Manipulate" onclick="handler_manipulateImage()">
61     </form>
62     <br><br><br>
63   </div>
64
65   <div id = "other_image"></div>
66
67   <br><br><br>
68   <div>
69     <br><br><br>
70     <input type="button" class="btn btn-success" value="Close" onclick="handler_tmp_cleaner()">
71   </div>

```

Figura 2.23: Página de Image Manipulation em html.

Destaca-se aqui o elemento “div” com id=“original\_image”, dentro dum outro “div” com class=“columnleft”, o elemento “div” com id=“manipulated\_image”, dentro dum “div” com class=“columnright”, e o elemento “div” vazio com id=“other\_image”.

Os dois primeiros “divs” servem para carregar e mostrar a imagem original, e para mostrar a imagem manipulada, respetivamente. O outro “div” é usado pelos métodos de manipulação para colocarem nele os “inputs” que forem precisos.

O primeiro método de manipulação de imagem que fizemos foi o “watermark”, que adiciona uma imagem noutra, e que coloca neste último “div” um outro “div”,

com class="columnleft", que em si terá um "input" de imagem, semelhante ao que se encontra no primeiro "div" com id="original\_image".

```

15 import {manipulateImage_unknown, sendFile_unknown, show_manipulated_image_unknown} from './image_manipulation_unknown.js'
16
17 var file;
18 window.handler_updatePhoto = function handler_updatePhoto(event) {
19     file = updatePhoto(event);
20 }
21
22 window.handler_tmp_cleaner = function handler_tmp_cleaner() {
23     tmp_cleaner();
24 }
25
26
27
28
29
30
31
32
33 window.handler_im_modificador = function handler_im_modificador(event) { //chamado quando se seleciona uma modifi
34
35     var modificador = document.getElementById("algo").value;
36     $("#other_image").empty(); //também limpa as cenas da div da da 2ª imagem, para evitar acumulador
37     $("#manipulated_image").empty(); //limpar o div onde fica a imagem manipulada
38     if (modificador == "Watermark") im_watermark();
39     //if (modificador == "Vignette") im_vignette();
40     if (modificador == "Bordador") im_inputs_bordador();
41     if (modificador == "Sepsis/Lomography") im_inputs_sepia_lomografia();
42     if (modificador == "Saturation") im_inputs_saturador();
43     if (modificador == "Gamma") im_inputs_gamma();
44     if (modificador == "Intensifier") im_inputs_intensificador();
45     if (modificador == "Resizer") im_inputs_resizer();
46 }
47
48
49
50 window.handler_manipulateImage = function handler_manipulateImage() { //chamado quando se clica no botão Manipule
51
52     var manipulation = document.getElementById("algo").value;
53     //alert(manipulation)
54     if (manipulation == "Watermark") manipulateImage_watermark();
55     if (manipulation == "Vignette") manipulateImage_vignette(file); //o watermark utiliza uma variável file declarada d
56     if (manipulation == "Bordador") manipulateImage_bordador(file);
57     if (manipulation == "Sepsis/Lomography") manipulateImage_sepia_lomografia(file);
58     if (manipulation == "Saturation") manipulateImage_saturador(file);
59     if (manipulation == "Gamma") manipulateImage_gamma(file);
60     if (manipulation == "Intensifier") manipulateImage_intensificador(file);
61     if (manipulation == "Resizer") manipulateImage_resizer(file);
62     if (manipulation == "Unknown") manipulateImage_unknown(file);
63 }
```

Figura 2.24: Script "image\_handler.js".

Devido à grande quantidade de métodos de manipulação que foram feitos (estão presentes 9 no "select" da imagem 2.23, mas alguns destes, como a "Sepsis/Lomography", o "resizer", e o "Unknown", contém na verdade 2, 4, 3 e métodos diferentes, respectivamente), decidiu-se criar um ficheiro javascript para definirem-se as funções necessárias para cada um desses métodos listados, funções essas que são posteriormente importadas para um ficheiro javascript chamado "image\_handler.js", que simplesmente delega as funcionalidades pretendidas para as funções que foram importadas. Isto evita que haja um único "hiper-script" com milhares de linhas de código de todas as funções precisas.

Quando um método é escolhido, os conteúdos das “divs” correspondentes à imagem manipulada e aos “inputs” específicos de cada método, são apagados, de modo a dar lugar aos novos conteúdos a serem gerados pelo método escolhido.

Devido à grande quantidade de métodos de manipulação de imagens que fizemos, explicitaremos apenas como implementámos 2 métodos: a “watermark”, e o “bordador”. Todos os métodos têm nomes pelos quais se pode inferir a manipulação que fazem, com a exceção propositada do “unknown”, cujo propósito é surpreender o utilizador com uma manipulação de imagem que este não espere (ou pelo menos, criar e satisfazer nele a curiosidade de saber o que este método faz ao certo).

Começemos pela “watermark”. Quando este método é escolhido, a “div” com id=“other\_image”, correspondente aos “inputs” específicos de cada método, tem os seus conteúdos inicialmente apagados, e é logo populada por um outro “div” (chamado “divs”), com class=“columnleft”, que fica logo abaixo do “div” da escolha de imagem original. Este novo “div” é em si populado por um “input” de imagem, que permite ao utilizador escolher a imagem que quer adicionar como marca de água à imagem original, e por um elemento “canvas”, que é usado para mostrar na página html a imagem escolhida (à imagem do que ocorre com o “div” com id=“original\_image” que está definido na página html).

Como já foi referido, o utilizador escolhe ambas as imagens do seu computador pessoal, para que não tenha que as carregar primeiro na galeria (o que poderia ser frustrante se no final não gostasse da imagem manipulada, se é que é possível manipulá-las).

Além disso, também é adicionado ao “div” com id=“other\_image” (e não ao “div” alinhado à esquerda que é “appended” nele) um outro “div” (chamado “divsFator”) com um “input” numérico, em que o utilizador tem de inserir um fator de transparência da imagem de marca de água (tem de ser um valor decimal entre 0 e 1, caso contrário, é apresentado um “alert” de erro).

Segue-se na próxima página o código da função que cria e distribui estes elementos.

```
114 function im_watermark() { //tinha um 'event' aqui dentro (depois da solução estar a fu
115
116     var divsFator = document.createElement("div");           //criar input para o utilizad
117     var textoFator = document.createElement("input");
118     textoFator.type = "number";
119     textoFator.step = "0.1";
120     textoFator.min = "0";
121     textoFator.max = "1";
122     textoFator.id = "watermark_texto";
123     textoFator.style="color: blue";
124
125     var labelFator = document.createElement("label");
126     labelFator.innerHTML = "Fator de transparência (0 <= f <= 1)";
127     labelFator.setAttribute("for", textoFator.id);
128
129
130     divsFator.appendChild(textoFator);
131     $("#other_image").append(labelFator);
132     $("#other_image").append(divsFator);
133
134 //a parte acima inclui os campos de texto que são precisos (além da imagem). Abaixo,
135
136     var divs = document.createElement("div");           //tudo funciona, menos a imagem ser
137     divs.setAttribute("class", "columnleft");
138
139     var linebr = document.createElement("br");
140     var header = document.createElement("h3");
141     header.appendChild( document.createTextNode("Imagen a inserir na imagem principal"))
142
143     var hr = document.createElement("hr");
144
145     var im_escolher = document.createElement("input");
146     im_escolher.type="file";
147     im_escolher.accept="image/*";           //im_escolher.setAttribute("accept", "image
148     im_escolher.onchange = updatePhoto2;    //a solução para meter a imagem (2º a da
149     | | | | | | | | | | | | | | | | | | //a imagem que aparece para meter como water
150
151     var im_canvas = document.createElement("canvas");   //cria o espaço onde vai ficar a
152     im_canvas.id = "photo2";
153     im_canvas.width="550";
154     im_canvas.height="450";
155
156     divs.appendChild(linebr);
157     divs.appendChild(header);
158     divs.appendChild(hr);
159     divs.appendChild(im_escolher);
160     divs.appendChild(im_canvas);
161
162     document.getElementById("other_image").appendChild(divs);
163 }
```

Figura 2.25: Função "im\_watermark".

As funções que permitem visualizar a imagem original escolhida e guardar os seus dados (“updatePhoto”), visualizar a imagem escolhida como marca de água e guardar os seus dados (“updatePhoto2”), bem como a função que ocorre ao clicar no botão submeter (que no método “watermark”, é a função “manipulateImage\_watermark”) e a função que mostra a imagem manipulada (caso ela seja manipulada com sucesso) na “div” alinhada à direita (função “show\_manipulated\_image”), são mostradas de seguida.

```
C: > Users > joaoa > OneDrive > Ambiente de Trabalho > Projeto Labi > projeto_com_parametros_no_
1  var file;           //serve para a função manipulateImage() poder saber se
2  var file2;
3
4  function updatePhoto(event) {                      //apresentar foto no 1
5      var reader = new FileReader();
6      reader.onload = function(event) {
7          //Create an imagem
8          var img = new Image();
9          img.onload = function() {
10             //Put imagen on screen
11             const canvas = $("#photo")[0];
12             const ctx = canvas.getContext("2d");
13             ctx.drawImage(img,0,0,img.width,img.height,0,0,550, 450);
14         }
15         img.src = event.target.result;
16     }
17
18     file = event.target.files[0];    //o event.target.files[0] representa
19     //Obtain the file               //a Blob, neste caso específico (e
20     reader.readAsDataURL(file);
21     return file;                 //dados que devem chegar ao servidor
22 }
23
24
25
26 function manipulateImage_watermark() {                //enviar
27     if(file != null && file2 != null) {
28        .sendFile_watermark(file, file2);
29     }
30     else alert("Image(s) missing!");
31 }
```

Figura 2.26: Funções "updatePhoto" e "manipulateImage\_watermark".

```

33  function.sendFile_watermark(file, file2) {
34
35
36      //obter valor de f e verificar se é válido
37      var f = document.getElementById("watermark_texto").valueAsNumber;
38      if( f == NaN || f < 0 || f > 1 || f == "" || f == undefined) {
39          alert("Por favor insira um número válido.");
40      }
41      else {
42          //o que é enviado
43          //os valores são
44
45          var data = new FormData();
46          data.append("file1", file);
47          data.append("file2", file2);
48          data.append("f", f);
49
50
51          var xhr = new XMLHttpRequest();
52          xhr.open("POST", "/watermark");
53
54
55          xhr.onreadystatechange = () => {
56              // Call a function when the state changes.
57              if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
58                  // Request finished. Do processing here.
59                  show_manipulated_image(JSON.parse(xhr.response));           //o JSON.
60              }
61          };
62
63          xhr.upload.addEventListener("progress", updateProgress(window), false);
64          xhr.send(data);
65
66
67      }
68  }
69
70  function updateProgress(evt){
71      if(evt.loaded == evt.total) alert("Submitted successfully.");
72  }

```

Figura 2.27: Função "sendFile\_watermark", que é chamada pela função mostrada acima "manipulateImage\_watermark".

```

75  function show_manipulated_image(response) {      //mostra a imagem manipulada; resultado das 2 outras imagens
76
77  |   document.getElementById("manipulated_image").innerHTML = "";           //limpar div para não ficar com v
78
79  |   if (response["result"] != "watermark_falhou") {
80  |       let water_image = document.createElement("img");
81  |       water_image.src = ".."+response["result"];           //depois de muitas horas a falhar no carregamento
82  |       water_image.width = "550";
83  |       water_image.height = "450";
84  |       document.getElementById("manipulated_image").appendChild(water_image);
85  |   }
86  |   else {
87  |       alert("The 2 chosen images could not be manipulated into a watermarked image.");
88  |   }
89  }
90
91
92  function updatePhoto2(event) {                  //apresentar foto que se pretenda inserir como watermark (q
93
94  |   var reader2 = new FileReader();
95  |   reader2.onload = function(event) {
96  |       //Create an imagem
97  |       var img = new Image();
98  |       img.onload = function() {
99  |           //Put imagen on screen
100 |           const canvas = $("#photo2")[0];
101 |           const ctx = canvas.getContext("2d");
102 |           ctx.drawImage(img,0,0,img.width,img.height,0,0,550, 450);
103 |       }
104 |       img.src = event.target.result;
105 |   }
106
107 |   file2 = event.target.files[0];
108 |   //Obtain the file
109 |   reader2.readAsDataURL(file2);
110
111 }

```

Figura 2.28: Funções "show\_manipulated\_image", e "uploadPhoto2".

Vejamos agora como é que os dados das imagens e do fator de transparência são usados no servidor, após serem enviados para este a partir do POST que se encontra implementado na função "sendFile\_watermark", mostrada na figura 2.27.

O método "watermark"começa por validar o fator de transparência. Se este não

representar um valor válido, é retornado um objeto json com uma mensagem de erro para a função "show\_manipulated\_image"(que só mostra a imagem caso seja retornado um objeto com um caminho válido da imagem manipulada no servidor).

Caso seja validado com sucesso o fator de transparência, o método guarda ambas

as imagens na pasta ./tmp, e verifica se a manipulação desejada é possível (isto é, verifica se a imagem da marca de água tem dimensões "height"e "width"menores que a imagem principal, através do método "watermark\_positions"importado do

ficheiro auxiliar). Se não for, é devolvido a mesma mensagem de erro acima. Se for, então o método procede para a manipulação da imagem (através dum método também chamado "watermark", mas que é proveniente do módulo correspondente ao ficheiro auxiliar, onde todos os métodos de processamento de imagem estão definidos), guardando-a também na pasta ./tmp (sempre com nomes de imagem sintetizados), e devolve o caminho dessa imagem para que a função "show\_manipulated\_image" possa criar um elemento "html" com atributo "src" igual a esse caminho no servidor, o que resulta na visualização da imagem na página html.

```

257     # watermark
258     @cherrypy.expose
259     def watermark(self, file1, file2, f):
260         #verificar se f é válido
261         try:
262             f=float(f)
263             assert(f>=0 and f<=1)
264         except:
265             print("Valor de f inválido")
266             result = "watermark_falhou"
267             return json.dumps({"result" : result}).encode("utf-8")
268
269
270         #guardar imagem 1 na pasta tmp do servidor
271         filename1 = baseDir + "/tmp/" + file1.filename
272         fileout = open(filename1, "wb")
273         while True:
274             data = file1.file.read(8192)
275             if not data: break
276             fileout.write(data)
277         fileout.close()
278
279
280         #guardar imagem 2
281         filename2 = baseDir + "/tmp/" + file2.filename
282         fileout = open(filename2, "wb")
283         while True:
284             data = file2.file.read(8192)
285             if not data: break
286             fileout.write(data)
287         fileout.close()
288
289
290         #obter coordenadas da imagem1 de modo a pôr toda a imagem2 no canto inferior direito da primeira (é retornado (None, None) se tal não for exequível)
291         start_x, start_y = imagem_modificador.watermark_positions(filename1, filename2)
292
293
294         #criar nova imagem a partir das 2 guardadas no servidor e guardar esta nova imagem com watermark na pasta tmp, para posterior GET no image_manipulation.js
295         cherrypy.response.headers["Content-Type"] = "application/json"
296         if start_x != None and start_y != None:
297             nome_imagem_servidor=imagem_modificador.watermark(filename1, filename2, f, start_x, start_y)
298             result = nome_imagem_servidor
299         else:
300             result = "watermark_falhou"
301             return json.dumps({"result" : result}).encode("utf-8")

```

Figura 2.29: Método "watermark".

```

9  def watermark_positions(fname1, fname2):
10     try:
11         im = Image.open(fname1)
12         width, height = im.size
13
14         im2 = Image.open(fname2)
15         width2, height2 = im2.size
16     except:
17         print("Não foi possível abrir os ficheiros através do PIL.Image.")
18         return (None, None)
19
20     x_start=width-width2           #para colocar as watermarks no canto inferior esquerdo
21     y_start=height-height2
22
23     if (x_start < 0) or (y_start < 0):
24         print("Não é possível meter a imagem2 na imagem1 sem que ocorra overlap.")
25         return (None, None)
26     return (x_start, y_start)

```

Figura 2.30: Método "watermark\_positions" do módulo "imagem\_modificador", importado do ficheiro auxiliar mencionado.

```

29  def watermark(fname1, fname2, f, start_x, start_y):
30
31     try:
32         im = Image.open(fname1)
33         new_im = im
34         im_water = Image.open(fname2)
35         im_water = im_water.convert("RGBA")
36         width_water, height_water = im_water.size
37     except:
38         print("Erro ao abrir imagens.")
39         return "watermark_falhou"
40
41     try:
42         for x in range(width_water):
43             for y in range(height_water):
44
45                 #p1 é um pixel da imagem original
46                 #p2 é um pixel da marca de água
47                 p1 = im.getpixel( (x+start_x, y+start_y) )
48                 p2 = im_water.getpixel( (x,y) )
49                 if(p2[3] == 0):
50                     continue
51
52                 r = int(p1[0]*(1-f)+p2[0]*f)
53                 g = int(p1[1]*(1-f)+p2[1]*f)
54                 b = int(p1[2]*(1-f)+p2[2]*f)
55
56
57             new_im.putpixel( (x+start_x, y+start_y), (r, g, b) )
58     except:
59         print("A colocação da watermark na imagem original falhou.")
60         return "watermark_falhou"
61
62
63     datetime = str(time.time_ns()//1000)
64
65     head, tail = os.path.split(fname1)
66     new_im_name = "tmp/"+tail+"-watermark-"+datetime+".png"
67
68     new_im.save(new_im_name)    #tive que alterar para .png, embora funcione
69     return new_im_name

```

Figura 2.31: Método "watermark" do módulo "imagem\_modificador".

As funções e métodos explícitos nas últimas 7 páginas, bem como as suas descrições, foram feitas de modo a implementar o método "watermark", e também para descrever a linha de pensamento seguida, bem como o fluxo de interações entre interface e servidor.

Como as descrições do funcionamento das interações entre funções javascript e métodos python e de como imaginámos e pensámos e implementámos as funcionalidades de processamento de imagem são semelhantes entre a manipulação de imagem "watermark" e as restantes, passaremos agora para uma exposição de código mais intensa.

Segue-se agora a exposição do código do método de manipulação de imagem "bordador", com o código da função "im\_bordador\_inputs", responsável por adicionar à página html os inputs necessários para que o servidor possa manipular a imagem conforme os desejos do utilizador.

```

61  function im_inputs_bordador() {
62      //criar input numérico (entre 0 e 255) para o parâmetro "diff"
63      var divs = document.createElement("div");
64      var textoDiff = document.createElement("input");
65      textoDiff.type = "number";
66      textoDiff.step = "1";
67      textoDiff.min = "0";
68      textoDiff.max = "255";
69      textoDiff.id = "diff_bordador";
70      textoDiff.style="color: blue";
71
72      var labelFator = document.createElement("label");
73      labelFator.innerHTML = "Pixel difference (for determining whether the difference
74      labelFator.setAttribute("for", textoDiff.id);
75
76      divs.appendChild(labelFator);
77      divs.appendChild(document.createElement("br"));
78      divs.appendChild(textoDiff);
79      divs.appendChild(document.createElement("br"));
80      divs.appendChild(document.createElement("br"));
81      divs.appendChild(document.createElement("br"));
82
83      //criar input True or False para o parâmetro "bw"
84      var select_bw = document.createElement("input");
85      select_bw.type = "checkbox";
86      select_bw.value = "on";
87      select_bw.id = "bw_bordador";
88
89      var label_bw = document.createElement("label");
90      label_bw.innerHTML = "Black and white bordered image, or just contoured image?";
91      label_bw.setAttribute("for", select_bw.id);
92
93      divs.appendChild(label_bw);
94      divs.appendChild(document.createElement("br"));
95      divs.appendChild(select_bw);
96
97
98      //dar append de ambos ao body da página html
99      document.getElementById("other_image").appendChild(divs);
100
101
102 }

```

Figura 2.32: Função "im\_bordador\_inputs".

```

1  function manipulateImage_bordador(file) {
2      if(file != null) {
3         .sendFile_bordador(file);
4      }
5      else alert("Image missing!");
6  }
7
8  function sendfile_bordador(file) {
9
10     var diff = document.getElementById("diff_bordador").valueAsNumber;
11     var bw = document.getElementById("bw_bordador").checked;
12
13     if( diff == NaN || diff < 0 || diff > 255 || diff == "" || diff == undefined) {
14         alert("Por favor insira um número válido.");
15     }
16     else {
17
18         var data = new FormData();
19         data.append("file", file)
20         data.append("diff", diff)
21         data.append("bw", bw)
22
23
24         var xhr = new XMLHttpRequest();
25         xhr.open("POST", "/bordador");
26
27
28         xhr.onreadystatechange = () => {
29             // Call a function when the state changes.
30             if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
31                 // Request finished. Do processing here.
32                 show_manipulated_image_bordador(JSON.parse(xhr.response));           //o JSON.parse é essencial para usar a resposta como dicionário, como é pretendido
33             }
34         };
35
36         xhr.send(data);
37
38     }
39 }
40

```

Figura 2.33: Função "manipulateImage\_bordador", invocada quando se clica no botão de submeter aquando este método "bordador" está selecionado, e função "sendfile\_bordador", que envia o POST ao servidor.

```

44  function show_manipulated_image_bordador(response) {    //mostra a imagem manipulada;
45
46      document.getElementById("manipulated_image").innerHTML = "";           //limpar div
47
48      if (response["result"] != "bordador_falhou") {
49          let image = document.createElement("img");
50          image.src = "./"+response["result"];           //depois de muitas horas a falhar
51          image.width = "550";
52          image.height = "450";
53          document.getElementById("manipulated_image").appendChild(image);
54      }
55      else {
56          alert("The chosen image could not be altered with Bordador.");
57      }
58  }

```

Figura 2.34: Função "show\_manipulates\_image\_bordador", responsável por interpretar o objeto json recebido do servidor e mostrar a imagem resultante, ou uma mensagem de erro, conforme o conteúdo do mesmo.

Do lado do servidor, relativamente à funcionalidade de manipulação de imagem "bordador":

```
327     @cherrypy.expose
328     def bordador(self, file, diff, bw):
329         #ler imagem e guardá-la na tmp
330         filename = baseDir + "/tmp/" + file.filename
331         with open(filename, "wb") as stream:
332             while True:
333                 data = file.file.read(29)
334                 if not data:
335                     break
336                 stream.write(data)
337
338         #verificar o valor da checkbox do bw
339         if bw == "true":
340             bw = True
341         else:
342             bw = False
343
344         #aplicar bordador
345         try:
346             diff=int(diff)
347             caminho_imagem = imagem_modificador.im_handler(filename, diff, bw)
348         except:
349             caminho_imagem = "bordador_falhou"
350
351         #devolver resultado (dizer apenas o caminho da nova imagem, se esta foi feita com sucesso, ou indicar que o método falhou)
352         return json.dumps({"result" : caminho_imagem}).encode("utf-8")
```

Figura 2.35: Método "bordador".

E o devido método proveniente do módulo "image\_modificador" mostrado na imagem acima, que é complexo (constituído por 3 métodos distintos), que marca as bordas na imagem recebida (enviada pela interface através do POST, com o parâmetro de diferença entre píxeis a considerar até a considerar como borda, bem como o parâmetro booleano de obter uma imagem em black and white, com as bordas a serem delimitadas numa cor, e os restantes píxeis noutra cor, ou de obter uma imagem em que apenas as bordas são delimitadas, retendo os restantes píxeis as suas qualidades já existentes).

```

134     #16) bordador
135     def is_edge(im, x,y, diff, bw):
136         #Obter o pixel
137         p = im.getpixel( (x , y) )
138         width, height = im.size
139
140         if x < width-1 and y < height-1 and x > 0 and y > 0:
141
142             #Vizinhos acima e abaixo
143             for vx in range(-1,1):
144                 for vy in [-1, 1]:
145                     px = im.getpixel( (x + vx, y + vy) )
146                     if abs(p[0]- px[0]) > diff:
147                         return (0,128,128)
148
149             #Vizinhos da esquerda e direita
150             for vx in [-1, 1]:
151                 px = im.getpixel( (x + vx, y) )
152                 if abs(p[0]- px[0]) > diff:
153                     return (0,128,128)
154
155             if bw :
156                 return (255,128,128)
157             else:
158                 return p
159
160
161     def iterativaDor(im, new_im, diff, bw):
162         width, height = im.size
163         for vx in range(1,width-1):
164             for vy in range(1,height-1):
165                 p_dor = is_edge(im, vx, vy, diff, bw)
166                 new_im.putpixel( (vx, vy), p_dor)
167
168     def im_handler(fname, diff, bw):
169
170         try:
171             im=Image.open(fname)
172
173             new_im=Image.new(im.mode, im.size)
174             #recursivaDor(im, new_im, 1, 1, diff, bw)      #os pixels
175             iterativaDor(im, new_im, diff, bw)
176         except:
177             print("Erro ao abrir imagem ou colocar as bordas nela.")
178             return "bordador_falhou"
179
180             datetime = str(time.time_ns()//1000)
181
182             head, tail = os.path.split(fname)
183             new_im_name = "tmp/"+tail+"-bordaDor-"+datetime+".png"
184
185             new_im.save(new_im_name)
186             return new_im_name

```

Figura 2.36: Métodos auxiliares do módulo "image\_modificador" para ajudar o método "bordador" no servidor a servir a imagem pretendida, modificada segundo os parâmetros pretendidos.

Vistas as implementações destes 2 métodos de manipulação de imagem ("watermark" e "bordador"), não mostraremos as implementações das restantes 7 (ou mais corretamente, 13 métodos restantes, tendo em conta as derivações que os métodos "sepsis/lomography", "resizer", e "unknown" contém).

Mostraremos agora, para terminar este capítulo, uma função e método de nome idêntico que são chamados quando a página de Image Manipulation é fechada, e que têm como objetivo limpar a pasta ./tmp.

```
168  function tmp_cleaner() {           //alertar o servidor de que pode eliminar os conteúdos temp
169    var xhr = new XMLHttpRequest();
170    xhr.open("POST", "/tmp_cleaner");
171    xhr.send();
172    //esperar por resposta do servidor
173    xhr.onreadystatechange = () => {
174      // Call a function when the state changes.
175      if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
176        window.close();           //faz o .js esperar que
177      }
178    }
179 }
```

Figura 2.37: Função "tmp\_cleaner", chamada quando a página Image Manipulation é fechada.

```
499  @cherrypy.expose
500  def tmp_cleaner(self):
501    print()
502    imagens_tmp = glob.glob(baseDir+"/tmp/*")
503    for imagem in imagens_tmp:
504      print("Ficheiro removido: "+imagem)
505      os.remove(imagem)          #demasiado
```

Figura 2.38: Método "tmp\_cleaner", chamado pelo POST enviado pela função de nome verossimilhante.

# Capítulo 3

## Resultados

A aplicação WEB que nos foi proposta visa gerir imagens; carregá-las, guardá-las, visualizá-las e manipulá-las. A aplicação WEB que desenvolvemos seguindo a metodologia descrita, permite o uso de todas essas funcionalidades.

Nós testámos e usufruímos da nossa aplicação WEB através da interface da aplicação, que pode ser usada a partir dum browser comum, inserindo o endereço ‘127.0.0.1:10013’ no mesmo.

Como já referido, decidimos implementar um mecanismo de registo/login antes de permitirmos o acesso de alguém às funcionalidades descritas. A imagem abaixo mostra o que se pode ver após a inserção do endereço mencionado acima no navegador:



Figura 3.1: Página de LogIn/Registro

Um utilizador pode criar uma conta (caso não tenha) ao preencher os campos “Username” e “Password” como pretender e clicar em “Register”, sendo que o nome de utilizador tem de ser único (é mostrado um alerta caso o utilizador se tente registar com um nome de utilizador já existente na base de dados).



Figura 3.2: Erro de Registo de Utilizador

Caso as credenciais inseridas sejam validadas com sucesso pelo servidor WEB, é mostrada ao invés uma mensagem a indicar esse sucesso.

Tendo a conta criada, o utilizador pode proceder para a página principal ao clicar em “Login” com as credenciais com que criou a conta.



Figura 3.3: Página Principal após login

Nesta nova página, o utilizador pode observar na barra de navegação opções que não conseguia ver antes: a Gallery Page, a UpLoad Page, e a Image Manipulation Page. As funcionalidades destas páginas já foram descritas na Introdução (Estrutura; organização visual) e também foram descritas extensivamente ao longo da Metodologia. Ao clicar numa destas opções, será apresentada ao utilizador, numa nova janela, uma página que permita ao mesmo fazer uso de funcionalidades que podem ser inferidas pelo nome da página.

Sem mais detalhes, visualizemos os resultados que se podem observar em cada página.

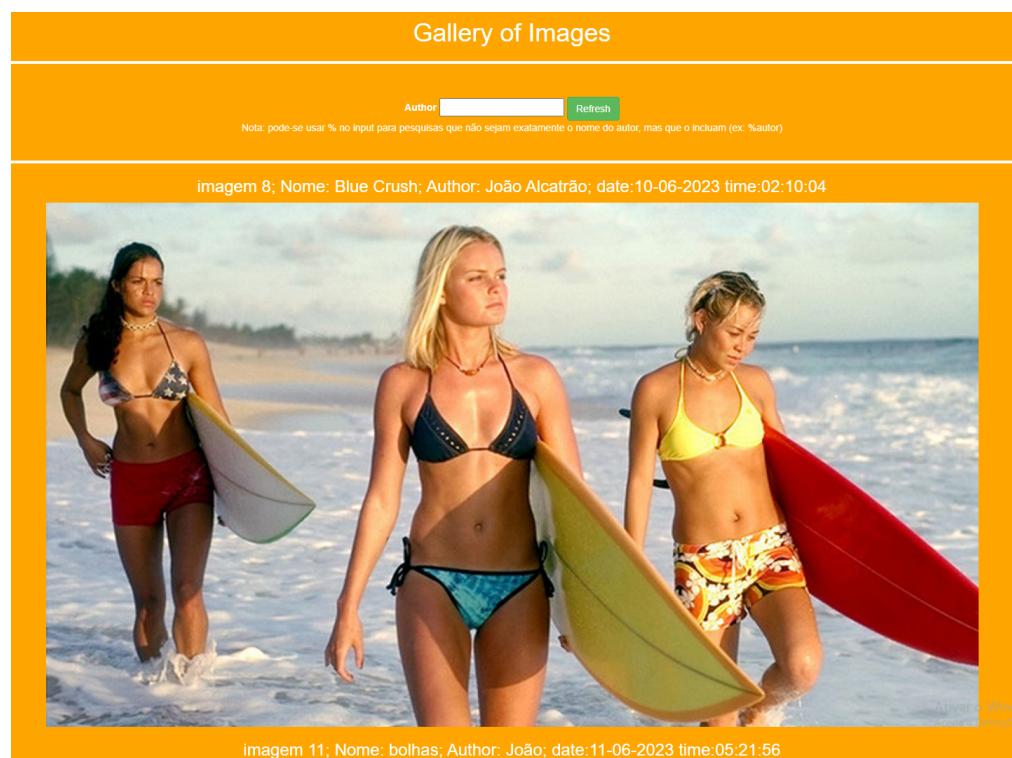


Figura 3.4: Gallery Page.

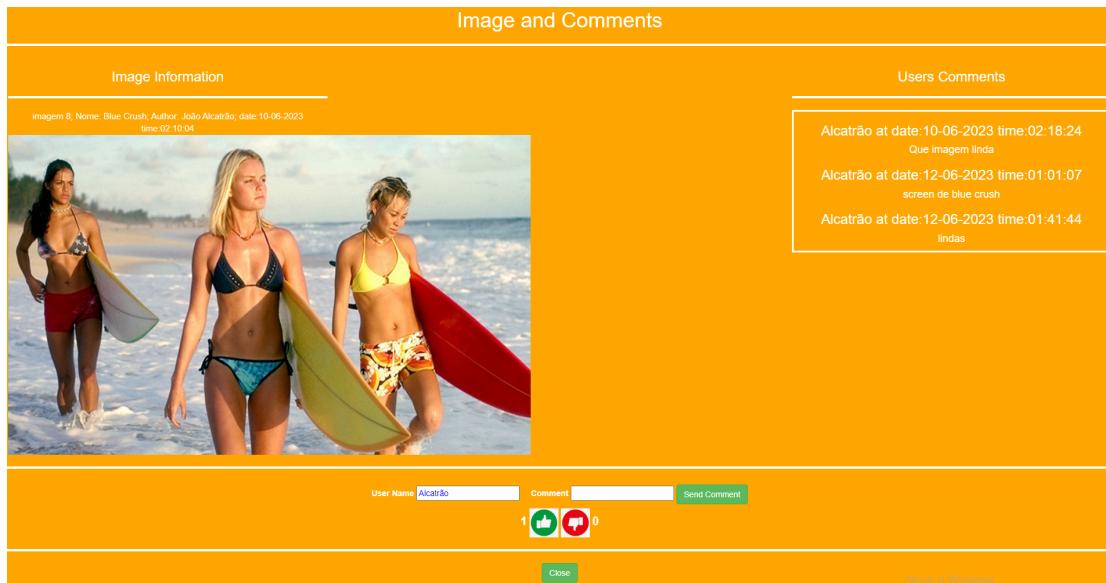


Figura 3.5: Image Page.

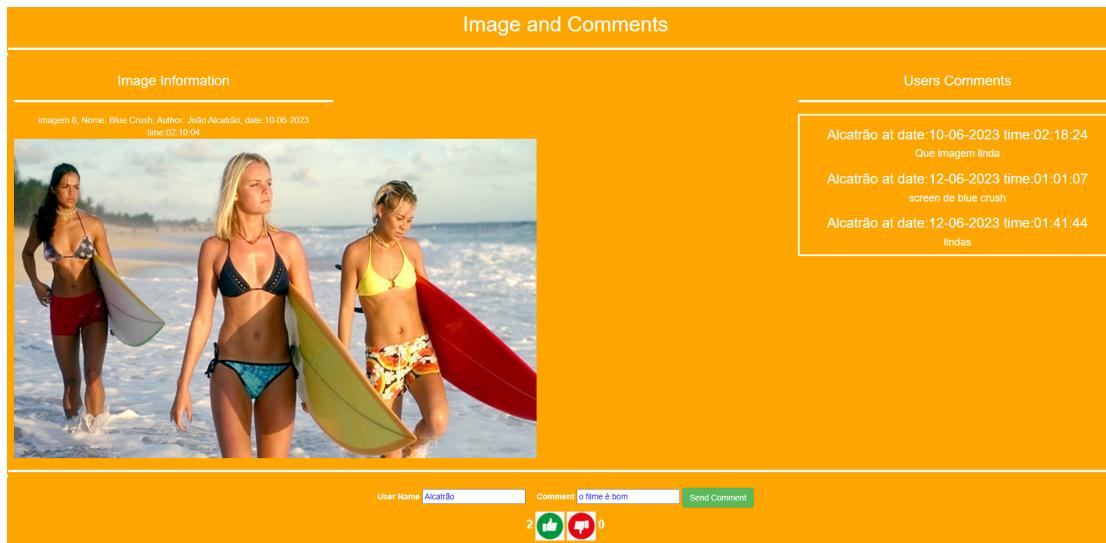


Figura 3.6: Image Page (escrevendo um comentário após fazer um gosto à imagem).

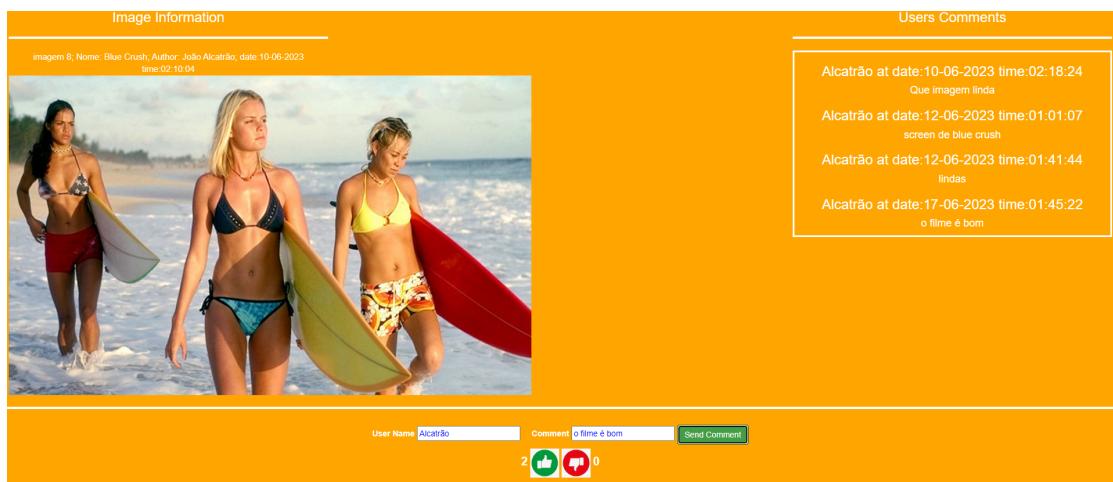


Figura 3.7: Image Page (após submeter o comentário).

imagem 12; Nome: espuma bolhas dor vermelha; Author: João Alcatrão; date:12-06-2023 time:01:45:08



Figura 3.8: Gallery Page (scroll mais para o fundo da página).

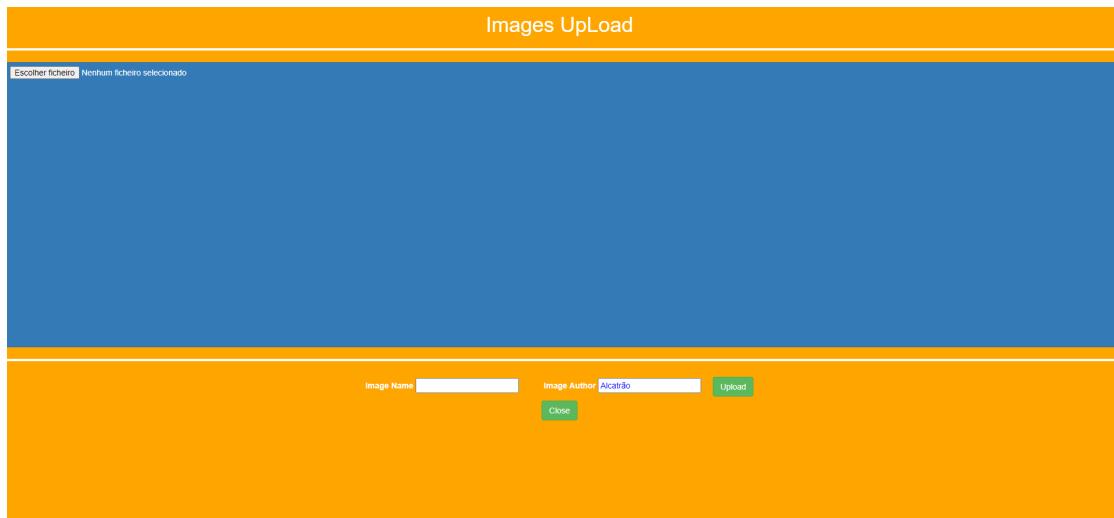


Figura 3.9: Upload Page.

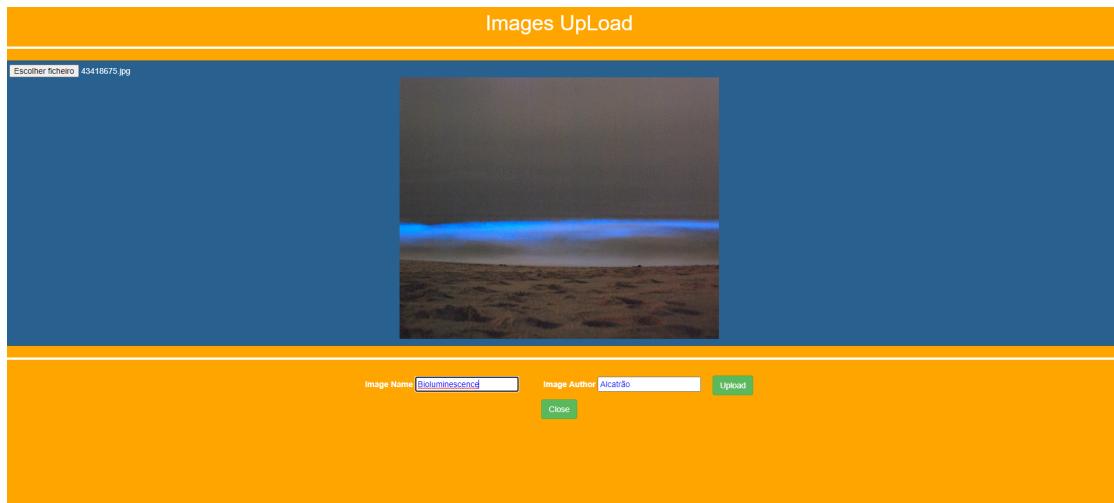


Figura 3.10: Upload Page (visualização de uma imagem escolhida, e do nome que lhe se quer dar).

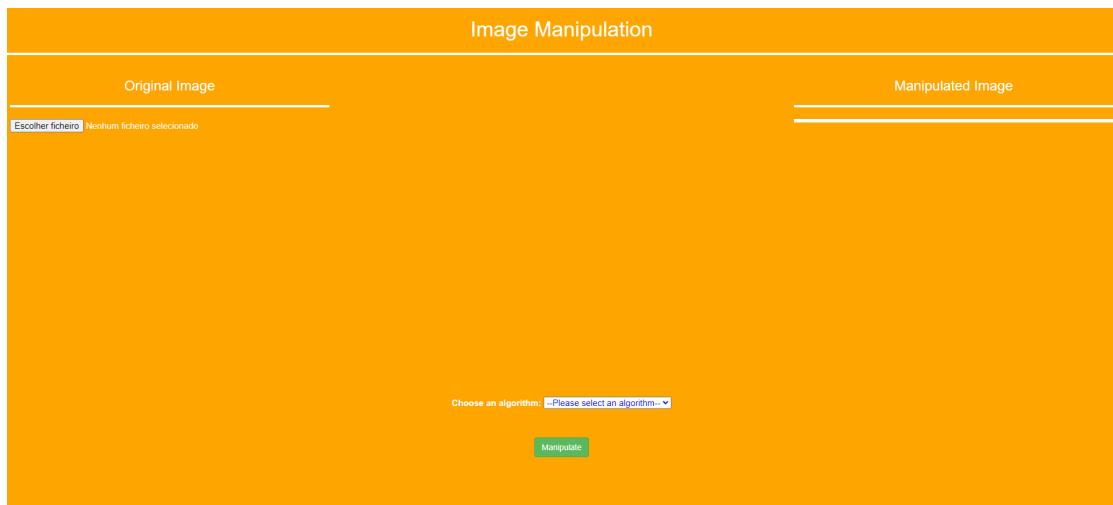


Figura 3.11: Image Manipulation Page.

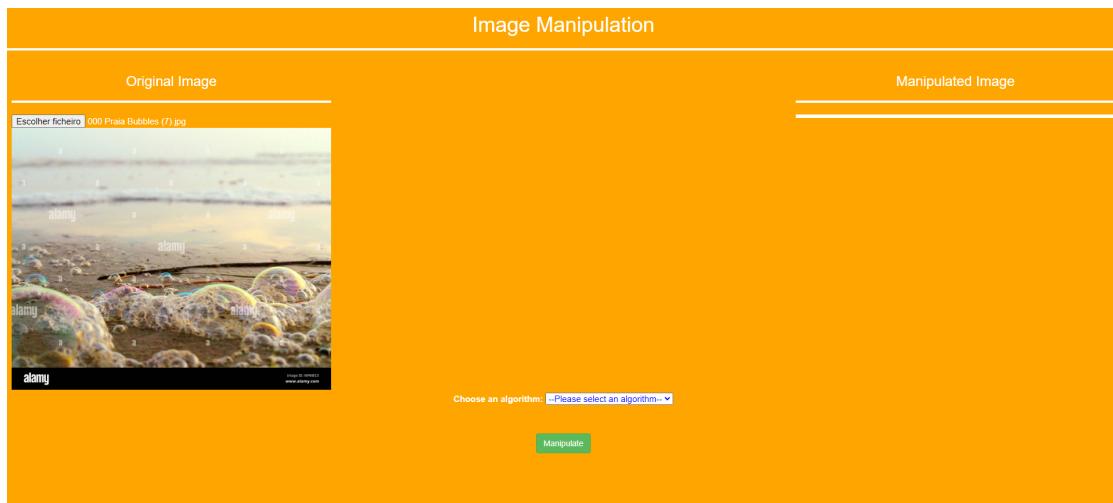


Figura 3.12: Image Manipulation Page (escolher uma imagem).

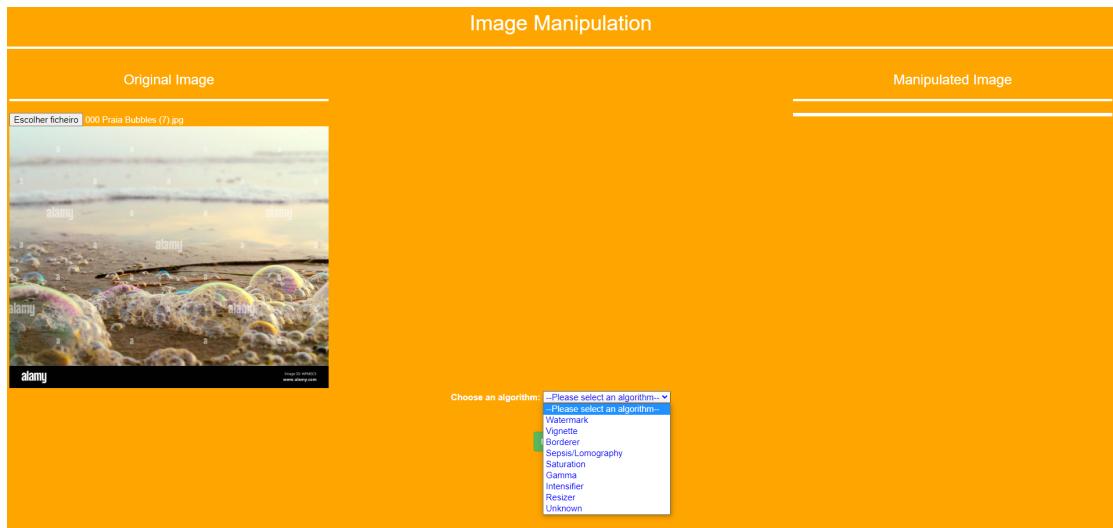


Figura 3.13: Image Manipulation Page (escolher um método).

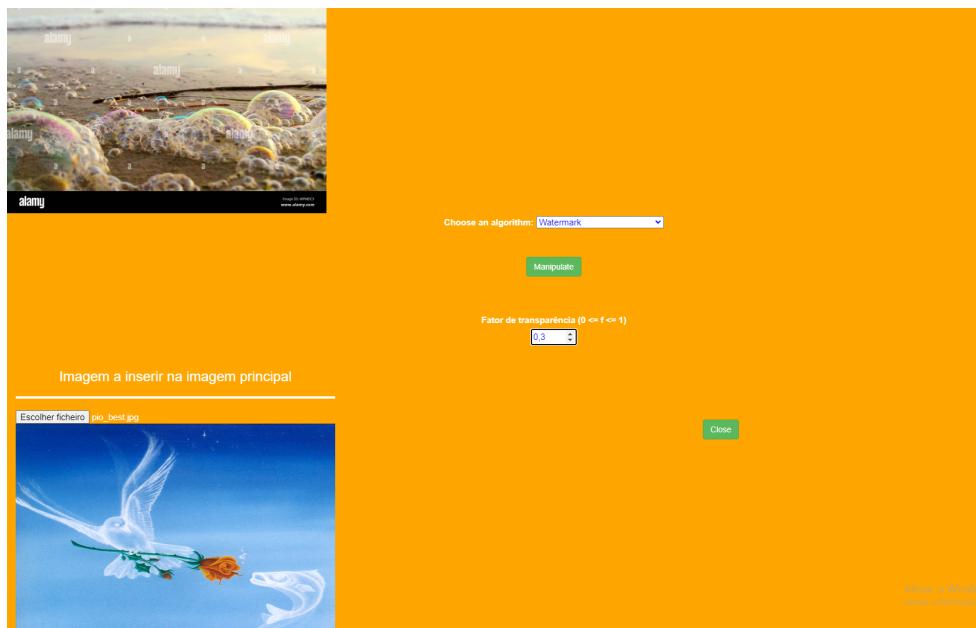


Figura 3.14: Image Manipulation Page (escolher outra imagem e meter um valor decimal no campo mostrado, como é pedido pelo método escolhido).

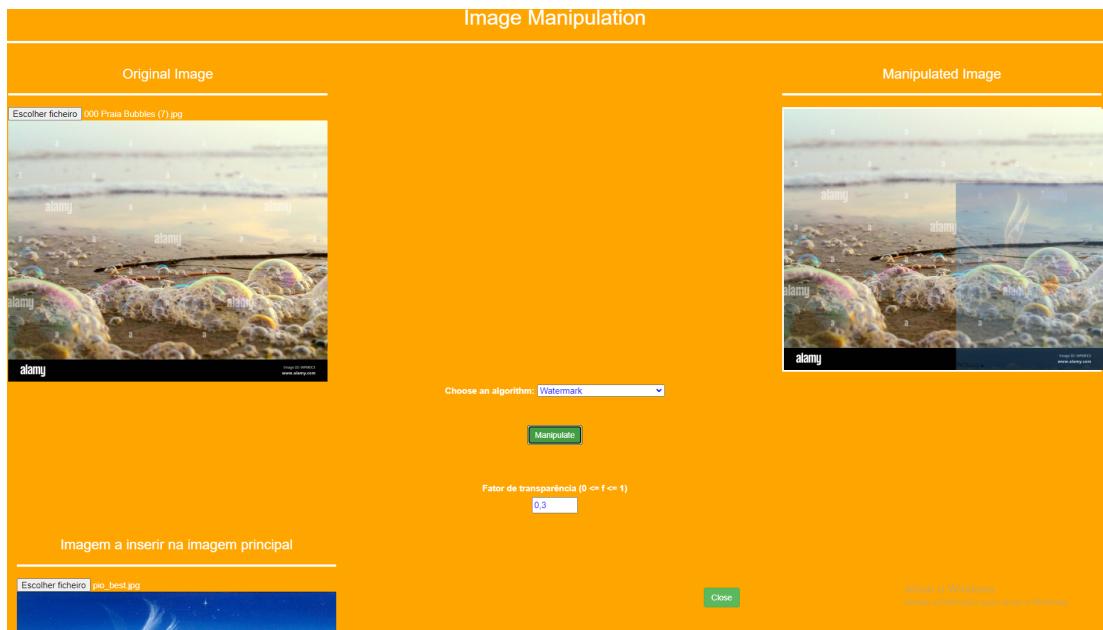


Figura 3.15: Image Manipulation Page (resultado da submissão das imagens escolhidas e do fator de transparência escolhido).

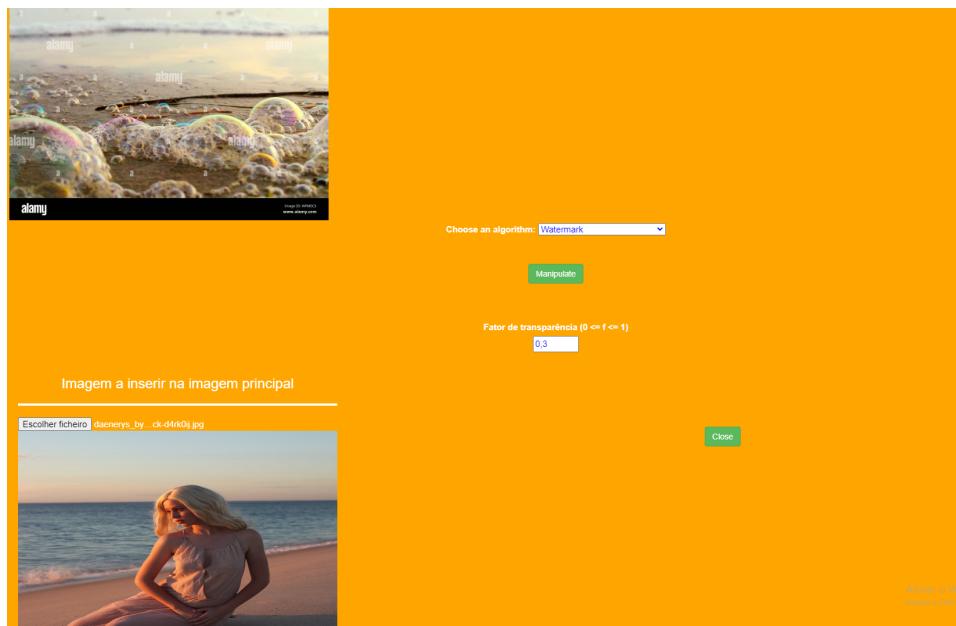


Figura 3.16: Image Manipulation Page (selecionar outra imagem secundária).

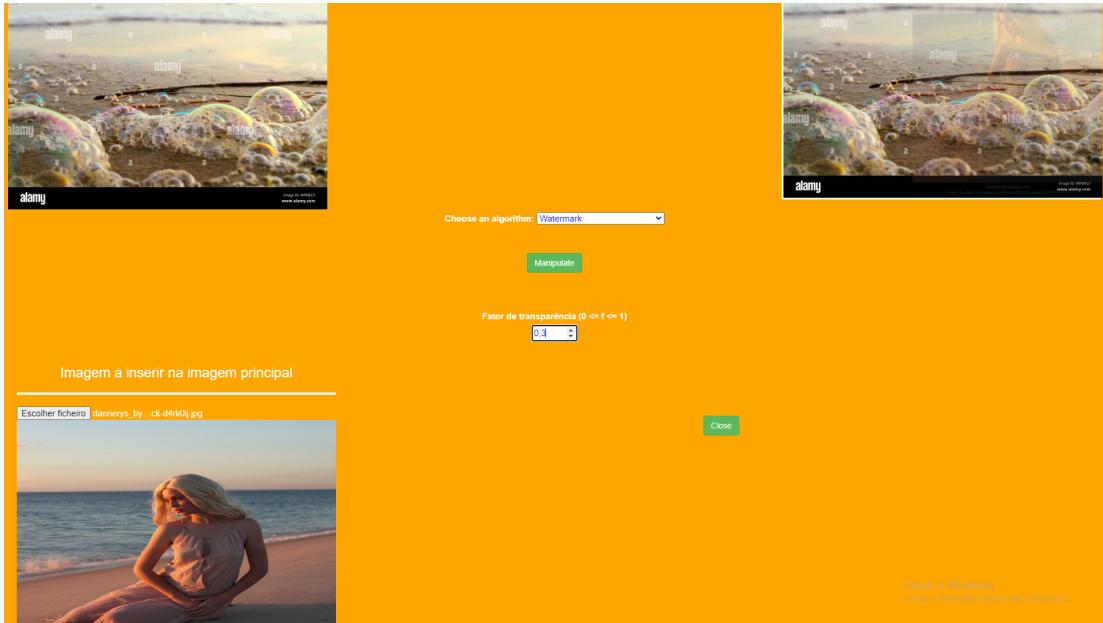


Figura 3.17: Image Manipulation Page (resultado da submissão com os novos parâmetros).

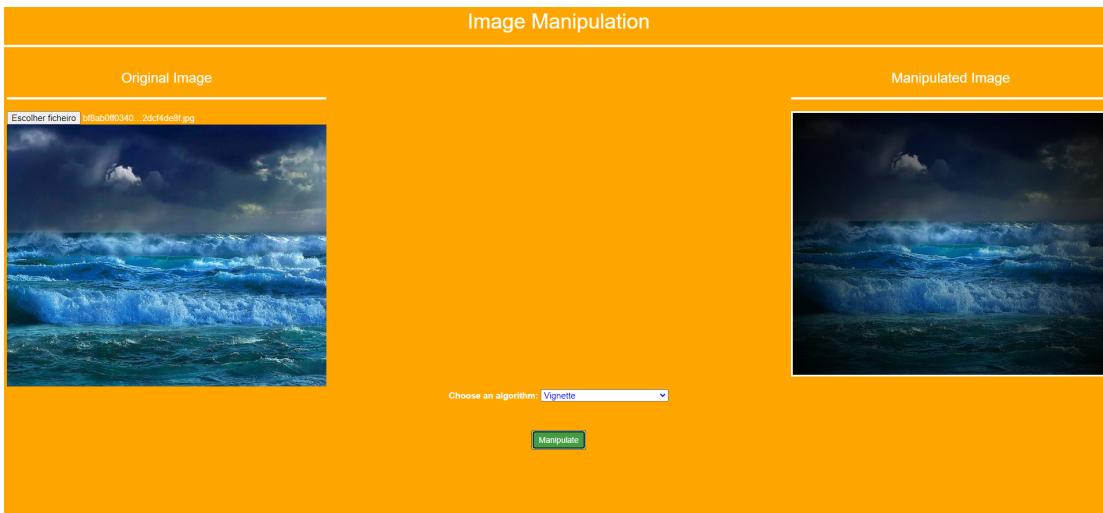


Figura 3.18: Image Manipulation Page (escolha de outra imagem e método de manipulação, e resultado da sua submissão ao servidor).

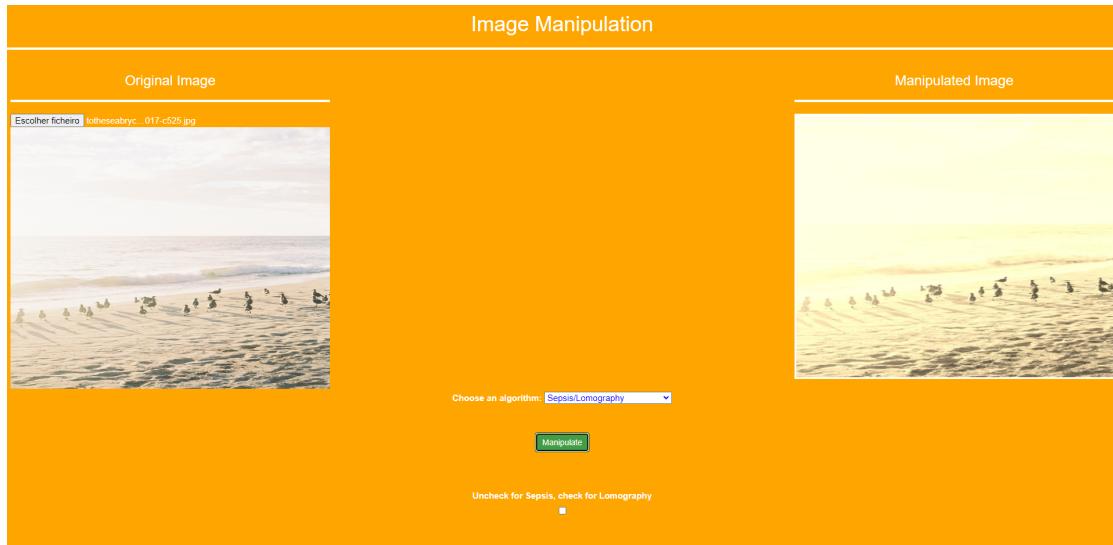


Figura 3.19: Image Manipulation Page (escolha de outra imagem e método de manipulação e da opção desejada, e resultado da sua submissão).

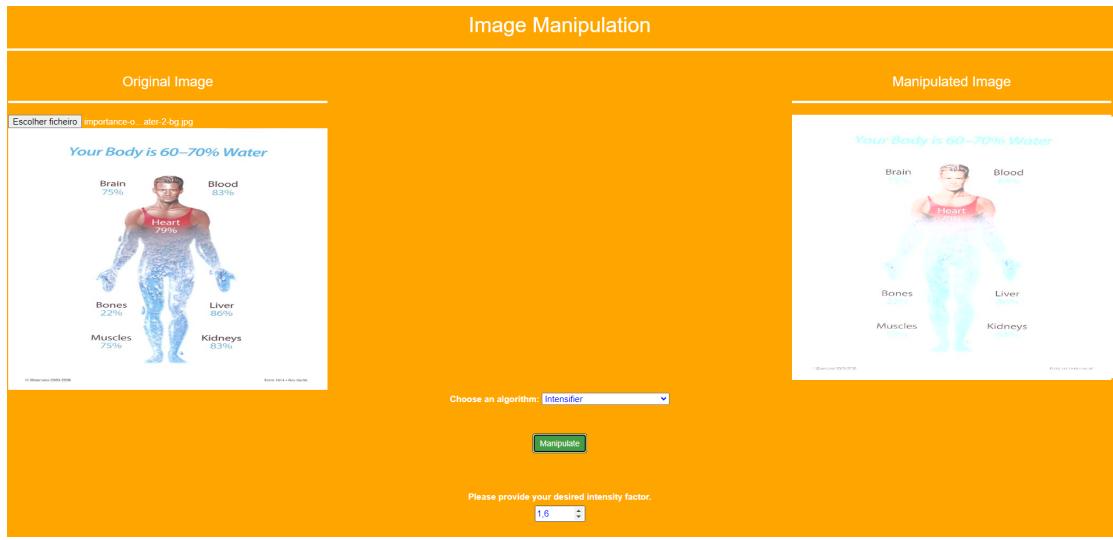


Figura 3.20: Image Manipulation Page (escolha de outra imagem e método de manipulação e de um valor no campo mostrado, e resultado da sua submissão).

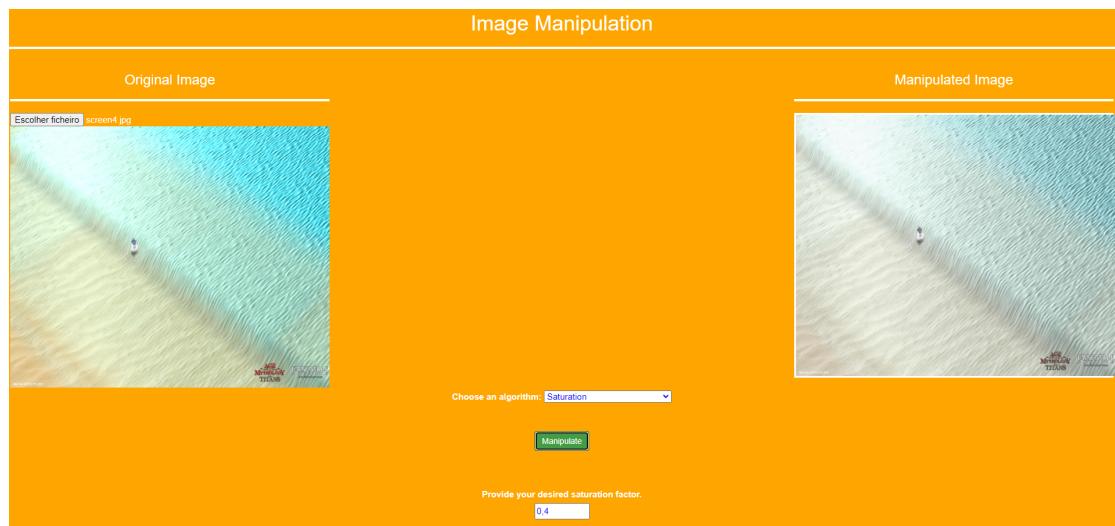


Figura 3.21: Image Manipulation Page (escolha de outra imagem e método de manipulação e do valor de saturação desejado, e resultado da sua submissão).

# Capítulo 4

## Análise

Analizando os resultados obtidos, das quais as images acima mostram o essencial do que a nossa aplicação consegue fazer, mas não tudo, podemos afirmar que conseguimos cumprir os objetivos que nos foram propostos, e conseguimo-lo usando boas práticas.

Estudando cada funcionalidade que a aplicação tinha de ter e que implementámos:

**Login/register:** a nossa aplicação tem um sistema de registo/login, que permite que apenas os utilizadores que criem conta na nossa aplicação possa usufruir totalmente dela (há livre acesso à página About e à própria página de registo/login apenas). Isto não serve apenas para frustrar os utilizadores, antes pelo contrário: permite que cada utilizador seja unicamente identificado pelo seu nome, e que mais nenhum outra possa fazer ações na aplicação sob o seu nome. O utilizador, ao criar conta e fazer login, verá que os campos de autor da página de carregamento de imagens e da página de apreciação e comentário de cada imagem já estarão preenchidos com o seu nome quando entrar nestas páginas.

O sistema pode parecer pouco ortodoxo, considerando que usamos um userID fabricado a partir da junção do nome de utilizador com a síntese da junção do nome de utilizador com a síntese da password para controlar o acesso dos utilizadores às diversas páginas da nossa aplicação. De qualquer modo, os objetivos de garantir a unicidade do utilizador na aplicação, e de garantir que a página de login/registo não possa ser facilmente ultrapassada adicionando uma expressão genérica e previsível no endereço, como "/html/upload", encontram-se cumpridos.

**Carregamento de imagens:** a nossa aplicação permite ao utilizador carregar imagens ao seu agrado na página Upload, separando a visualização do carregamento da mesma (o que permite ao utilizador alterar a sua escolha de imagem sem a carregar no sistema) e permite-o dar o nome que pretender que essas imagens apresentem na galeria do servidor.

A data e a hora do carregamento da imagem também são registadas pelo servidor na base de dados, e pode-se ver estas observações na galeria, e na página de apreciação (comentários e gostos) da respetiva imagem. Além disso, fomos além e garantimos que ninguém, a não ser o próprio utilizador em questão, possa submeter imagens sob o nome do próprio. A sua identidade é preservada, protegida e mantida.

**Visualização de imagens:** a nossa aplicação permite ao utilizador usar a página Gallery para ver uma lista de todas as imagens presentes no sistema, ordenadas por ordem ascendente do nome, permitindo a sua visualização. As imagens em si estão armazenadas no sistema de ficheiros da aplicação Web (renomeadas de acordo com a sua síntese SHA256 para evitar que imagens essencialmente diferentes mas com nomes idênticos se substituam), mas as informações que as acompanham (nome que o utilizador que a carregou lhe deu, o nome desse autor, e a data e hora de carregamento) são registadas na base de dados.

Além disso, é possível filtrar as imagens mostradas pelo autor, sendo possíveis tanto filtragens parciais como absolutas, como é indicado na página. Esta filtragem baseia-se em pesquisas na base de dados, pelo que tivemos o cuidado de usar boas práticas a escrever o nosso código para evitar SQL injection.

**Apreciação de imagem:** quando uma imagem é selecionada na galeria, é aberta uma nova página autónoma que apresenta todas as suas propriedades (autor, nome, hora e data de upload no sistema, comentários efetuados pelos utilizadores, e os gostos/desgostos deixados pelos mesmos). Esta página permite ao utilizador fazer os seus próprios comentários de forma exclusiva, com a garantia de que mais ninguém pode comentar sob o seu nome (tal como ocorre no carregamento de imagens).

Por último, é permitido ao utilizador facilmente fazer gostos/desgostos com um simples clique, e permite também retirá-los de igual forma com um clique (clicar duas vezes seguidas no botão gosto, adiciona na primeira instância um gosto à imagem, e retira na segunda esse gosto à imagem, tal como é o comportamento observado em funcionalidades semelhantes de algumas aplicações mais populares, como o Facebook e o Instagram. Isto é feito ao inserir e remover registo na base de dados, em especial os registo pertencentes ao utilizador que fez/desfez esses gostos/desgostos, garantindo assim consistência na aplicação, e garantindo que um utilizador que faça um gosto, feche e volte a abrir a página, e que volte a fazer gosto, acaba por desfazer o gosto que tinha inicialmente feito).

**Manipulação de imagens:** a nossa aplicação contém, na página de manipulação de imagens, uma vasta gama de métodos de processamento de imagens, bem como opções dentro de cada método que permitem ao utilizador manipular imagens como realmente desejar e pretender. Para facilitar a comparação entre a imagem original e a manipulada, a página mostra ambas lado a lado, e permite ao utilizador guardar a imagem manipulada no seu computador (de onde ele também pode carregar a imagem que deseja manipular).

Os diversos métodos de manipulação de imagem têm um conjunto de verificações aos parâmetros escolhidos que são feitas tanto do lado da interface, em javascript, como no lado do servidor, em python. As verificações do lado da interface servem mais para facilitar o processo de manipulação de imagem ao utilizador, enquanto que as verificações do lado do servidor servem para assegurar o bom funcionamento e segurança do sistema o quanto melhor que consiga.

Para terminar esta secção, apresentamos agora alguns dos testes unitários e funcionais que fizemos para assegurar o bom funcionamento do sistema, e garantir que este continue a correr em segurança, mesmo no caso de receber parâmetros e dados questionáveis. Fizemos um total de 43 testes, alguns com parâmetros válidos, a maioria com parâmetros inválidos, e garantimos que os nossos métodos passam neles todos.

```

c:\Users\j...pago\OneDrive - Ambiente de Trabalho\Projeto Lab\projeto.com.garamento.com.academy.seguro\projeto-final-lab2022q1\projeto\test_web_severify.py
1 import pytest
2 import imagem_modificador
3
4
5 #inputs para a função watermark: imagename1, imagename2, transparency_factor, start_x, start_y
6 def test_watermark_f():
7
8     #teste com parametro de transparência inválido
9     result = imagem_modificador.watermark("uploads/0495574b3b6221ce0af9d9df45571df603c2c5981ccc95d59c924c8779fe2ca5.jpg", "uploads/d519f0ef6f1e5a0f2625d19ff741697da66c522de4ff38ae542375c4b4dddb1.jpg", "letra", 0
10    expected_error = "watermark_falhou"
11    assert(result == expected_error)
12
13
14 def test_watermark_filesize():
15
16     #teste com parametro da segunda imagem errado
17     result = imagem_modificador.watermark("uploads/0495574b3b6221ce0af9d9df45571df603c2c5981ccc95d59c924c8779fe2ca5.jpg", 29, 0.2, 0, 0)
18    expected_error = "watermark_falhou"
19    assert(result == expected_error)
20
21
22 def test_watermark_xy():
23
24     #teste com parametros de posição da watermark que saem fora da imagem original
25     result = imagem_modificador.watermark("uploads/0495574b3b6221ce0af9d9df45571df603c2c5981ccc95d59c924c8779fe2ca5.jpg", "uploads/d519f0ef6f1e5a0f2625d19ff741697da66c522de4ff38ae542375c4b4dddb1.jpg", 0.2, 1000,
26    expected_error = "watermark_falhou"
27    assert(result == expected_error)
28
29
30 def test_watermark():
31
32     #teste com parametros válidos (a imagem resultante fica na pasta tmp; as 2 imagens original e watermark só são reescritas na mesma pasta onde estão, que é ./uploads, quando esta função é chamada na função watermark)
33     result = imagem_modificador.watermark("uploads/0495574b3b6221ce0af9d9df45571df603c2c5981ccc95d59c924c8779fe2ca5.jpg", "uploads/d519f0ef6f1e5a0f2625d19ff741697da66c522de4ff38ae542375c4b4dddb1.jpg", 0.2, 0, 0)
34    expected_result_start = "tmp"
35    assert(result[:4] == expected_result_start)
36
37
38 #inputs de vignette: imagename1, x_ref, y_ref
39 def test_vignette_filename():
40
41     #teste com nome de imagem inválido
42     result = imagem_modificador.vignette("don", 0, 0)
43    expected_error = "vignette_falhou"
44    assert(result == expected_error)
45
46
47 def test_vignette_x_y():
48
49     #teste com parametros posicionais errados
50     result = imagem_modificador.vignette("uploads/0495574b3b6221ce0af9d9df45571df603c2c5981ccc95d59c924c8779fe2ca5.jpg", -1000, "a")
51    expected_error = "vignette_falhou"
52    assert(result == expected_error)
53
54
55

```

Figura 4.1: Testes unitários aos métodos de manipulação de imagens.

```
121     def test_saturation_f():
122         #teste com parâmetro de fator inválido
123         with pytest.raises(SystemExit) as e:
124             |   imagem_modificador.saturation("images/img.jpg", "1a")
125             assert e.type == SystemExit
126             assert e.value.code == 29
127
128
129     def test_saturation():
130         #teste com parâmetros válidos
131         result = imagem_modificador.saturation("images/img.jpg", 2)
132         expected_result_start = "tmp/"
133         assert(result[0:4] == expected_result_start)
134
135
136
137     #inputs de gamma: imagename, fator_de_gamma
138     def test_gamma_modificaDor_imagename():
139         #teste com parâmetro de caminho de imagem inválido
140         result = imagem_modificador.gamma_modificaDor("0", 0)
141         expected_error = "gamma_falhou"
142         assert(result == expected_error)
143
```

Figura 4.2: Continuação de testes unitários aos métodos de manipulação de imagens.

```

C:\> Users > joaoa > OneDrive > Ambiente de Trabalho > Projeto Labi > projeto_com_parametros_nos_enderecos_seguro > projeto-final-labi2023g13 > project > test_web_server_requests.py > ...
1 import pytest
2 import sqlite3 as sql
3 from app import Root
4 from subprocess import Popen
5 import time, requests
6 import cherrypy, json, hashlib
7 from cherrypy.test import helper
8
9
10 #testes persistência
11
12 def test_database_select_votes():
13     db = sql.connect("database.db")
14     result = db.execute("SELECT * FROM votes")
15     linhas = result.fetchall()
16     db.close()
17
18     assert(linhas[0])==(79, 8, 2, 1, 0)           #o primeiro registo na tabela votos da base de dados é: 79|8|2|1|0 (id, imgid, author, ups, downs)
19
20
21 def test_database_select_comments():
22     db = sql.connect("database.db")
23     result = db.execute("SELECT * FROM comments")
24     linhas = result.fetchall()
25     db.close()
26
27     assert(linhas[1][3])=="Que imagem linda"      #o segundo registo na tabela dos comentários tem o comentário "Que imagem linda"
28
29
30 def test_database_select_first_account():
31     db = sql.connect("database.db")
32     result = db.execute("SELECT * FROM accounts WHERE id=?", (1,))
33     linhas = result.fetchone()
34     db.close()
35
36     assert(linhas[1])=="Alcatrão"                  #a primeira conta registada na tabela de dados é a minha
37
38
39 def test_database_select_images():
40     db = sql.connect("database.db")
41     result = db.execute("SELECT * FROM images WHERE author=?", ('João Alcatrão',))
42     linhas = result.fetchall()
43     db.close()
44
45     assert(len(linhas)) == 5                      #dei upload a 5 imagens com este nome de autor
46
47
48
49 def test_database_insert_existent_account():
50     with pytest.raises(sql.IntegrityError) as e:
51         db = sql.connect("database.db")
52         db.execute("INSERT INTO accounts(username, password, user_id) VALUES (?, ?, ?)", ("João", "password", "hash do nome mais a password encriptada"))
53         assert str(e.value) == 'UNIQUE constraint failed: accounts.username'    #a conta já existe, pelo que este erro deve ocorrer

```

Figura 4.3: Testes de persistência.

```
58 #teste funcional com uso de requests para aceder à página inicial
59 def test_pagina_index():
60     proc = Popen("exec python3 app.py", shell=True)          #sem
61     time.sleep(3)
62
63     servurl = "http://127.0.0.1:10013/"
64     r = requests.get(servurl)
65     time.sleep(1.5)
66
67
68
69     proc.terminate()                                     #ter
70
71     paginaEsperada = ''
72     with open("index.html" , 'r') as stream:
73         paginaEsperada = stream.read()
74
75
76     assert(r.text == paginaEsperada)
```

Figura 4.4: Teste funcional à aplicação com uso de requests.

```

122 #teste com registo, login e acesso à pagina da gallery
123 def test_register_login_and_access_gallery():
124     proc = Popen("exec python3 app.py", shell=True)           #sem o exec, o Popen spawnna
125     time.sleep(3)
126
127     servurl = 'http://127.0.0.1:10013/acios/doRegister?username=test&password=test'
128     r_register = requests.get(servurl)
129     time.sleep(1.5)
130
131     servurl = 'http://127.0.0.1:10013/acios/doLogin?username=test&password=test'
132     r_login = requests.get(servurl)
133     time.sleep(1.5)
134
135     servurl = 'http://127.0.0.1:10013/acios/doLogin?username=test&password=test'
136     r_login = requests.get(servurl)
137     time.sleep(1.5)
138
139
140
141     h = hashlib.sha256()                                     #calcular o userID
142     h.update("test".encode())
143     password = h.hexdigest()
144
145
146     s = hashlib.sha256()
147     s.update(password.encode())
148     s.update("test".encode())
149     userID = "test"+s.hexdigest()
150
151
152     servurl = 'http://127.0.0.1:10013/page_gallery?userID=' +(userID
153     r_gallery = requests.get(servurl)
154     time.sleep(1.5)
155
156
157     proc.terminate()
158
159
160     db = sql.connect("database.db")
161     db.execute("DELETE FROM accounts WHERE username = ?", ("test",))
162     db.commit()
163     db.close()
164
165
166
167
168     assert(r_register.json() == {"result" : "Conta criada."})
169     assert(r_login.json() == {"result" : "html/inicio.html", "userID": userID})
170
171     pagina_galeria=''
172     with open("html/gallery.html", 'r') as stream:
173         pagina_galeria = stream.read()
174
175     assert(r_gallery.text == pagina_galeria)

```

Figura 4.5: Teste funcional à aplicação com registo, login, e acesso à página da galeria.

```

122 #teste com registo, login e acesso à pagina da gallery
123 def test_register_login_and_access_gallery():
124     proc = Popen("exec python3 app.py", shell=True)           #sem o exec, o Popen spawnna
125     time.sleep(3)
126
127     servurl = 'http://127.0.0.1:10013/acios/doRegister?username=test&password=test'
128     r_register = requests.get(servurl)
129     time.sleep(1.5)
130
131     servurl = 'http://127.0.0.1:10013/acios/doLogin?username=test&password=test'
132     r_login = requests.get(servurl)
133     time.sleep(1.5)
134
135     servurl = 'http://127.0.0.1:10013/acios/doLogin?username=test&password=test'
136     r_login = requests.get(servurl)
137     time.sleep(1.5)
138
139
140
141     h = hashlib.sha256()                                     #calcular o userID
142     h.update("test".encode())
143     password = h.hexdigest()
144
145     s = hashlib.sha256()
146     s.update(password.encode())
147     s.update("test".encode())
148     userID = "test"+s.hexdigest()
149
150
151     servurl = 'http://127.0.0.1:10013/page_gallery?userID=' +(userID
152     r_gallery = requests.get(servurl)
153     time.sleep(1.5)
154
155
156
157     proc.terminate()
158
159
160     db = sql.connect("database.db")
161     db.execute("DELETE FROM accounts WHERE username = ?", ("test",))
162     db.commit()
163     db.close()
164
165
166
167
168     assert(r_register.json() == {"result" : "Conta criada."})
169     assert(r_login.json() == {"result" : "html/inicio.html", "userID": userID})
170
171     pagina_galeria=''
172     with open("html/gallery.html", 'r') as stream:
173         pagina_galeria = stream.read()
174
175     assert(r_gallery.text == pagina_galeria)

```

Figura 4.6: Teste simples aos métodos de acesso a páginas do servidor.

```
===== test session starts =====
platform linux -- Python 3.10.6, pytest-6.2.5, py-1.10.0, pluggy-0.13.0
rootdir: /mnt/c/users/joaoa/onedrive/ambiente de trabalho/Projeto Labi/projeto_com_parametros_nos_enderecos_seguro/proje
to-final-labi2023g13/project
collecting 0 items
collected 43 items

test_web_server.py ..... [ 74%]
test_web_server_requests.py .. [100%]

===== 43 passed in 59.72s =====
```

Figura 4.7: Resultado da execução dos testes.

# Capítulo 5

## Conclusões

Todos os objetivos forma cumpridos, todas as funcionalidades foram implementadas, e estamos imensamente satisfeitos com os resultados da nossa aplicação. Tendo em conta tudo o que foi demonstrado ao longo deste relatório, podemos afirmar que a nossa solução é complexa e foi muito trabalhada. Aprofundámos agudamente os nossos conhecimentos em Javascript e Cherrypy, através de programação exaustiva nesta linguagem e framework, e solidificámos as nossas competência em programação Python, em SQL, tivemos novas experiências com HTML e CSS e LaTeX, e aplicámos métodos aprendidos ao longo da disciplina relativos a criptografia e testagem.

Desejamos terminar este relatório afirmando que estamos satisfeitos com o trabalho que fizemos, com a perseverânciia que demonstrámos para com nós próprios, e sentimos-nos felizes com a aplicação criada, a nossa bela aplicação.

# Contribuições dos autores

João Alcatrão - interface (html, css, javascript), servidor e métodos de manipulação de imagem (cherrypy, python), base de dados (sql), testes unitários e funcionais, relatório

José Jordão - interface (html, css), servidor e métodos de manipulação (cherrypy, python), relatório (latex)

Mónica Pereira - servidor (cherrypy, python), base de dados (sql), relatório

Pedro Costa - relatório (latex)

João Alcatrão José Jordão Mónica Pereira Pedro Costa:

25%

25%

25%

25%

# GitHub e XCOA

GitHub:

<https://github.com/detiuaveiro/projeto-final-labi2023g13>

XCOA:

<https://xcoa.av.it.pt/labiproj13>