Projeto Final

UNIVERSIDADE DE AVEIRO

Pedro Ferreira, João Gaspar, Guilherme Santos,
Beatriz Oliveira

Projeto Final

DETI

UNIVERSIDADE DE AVEIRO

Pedro Ferreira, João Gaspar, Guilherme Santos, Beatriz Oliveira
(98620) pedrodsf@ua.pt, (107708) j.gaspar@ua.pt,
(107961) gssantos@ua.pt, (108606) beatrizroliveira@ua.pt

CodeUA: `https://code.ua.pt/projects/labi2022g10`

Aveiro, julho 2022

Resumo

Este relatório consiste na descrição do enquadramento das tecnologias usadas, na descrição da implementação efetuada, na apresentação de provas que demonstram o seu funcionamento correto, na análise à estrutura da aplicação e nas conclusões finais sobre a solução implementada.

Ao longo do relatório, é analisado e explicado o código do programa e do website que o acompanha, expondo todas as funções e características do projecto. Após isto, é efectuada uma demonstração do funcionamento do programa e do modo correto de interacção com a interface do mesmo e o relatório é concluído com algumas reflexões sobre o projeto e o contributo que este teve para o desenvolvimento das nossas capacidades. O repositório utilizado para o desenvolvimento da aplicação está alojado no CodeUA e o seu nome é "labi2022g10".

Agradecimentos

Durante a realização deste projeto obtivemos ajudas diretas e indiretas de várias pessoas, assim sendo gostariamos de expressar o nosso enorme agradecimento. Aproveitamos para agradecer ao **Professor Doutor António Manuel Adrego da Rocha** e ao **Professor Doutor Diogo Nuno Pereira Gomes** por nos terem proprocionado as bases necessárias para realizar este projeto (disponibilização de PowerPoints e Guiões que envolvem a matéria).

Conteúdo

Capítulo 1

Introdução

O objetivo deste projeto é desenvolver uma aplicação *Web Online* que armazena e visualiza imagens originais de forma segura e que faz o acompanhamento das transações digitais entre os colecionadores. A aplicação principal tem como módulo principal o *Cherrypy*, que responde corretamente aos pedidos realizados pelo utilizador da aplicação.

O relatório está dividido em 5 capítulos. Depois deste Capítulo 1 (Introdução), encontra-se o Capítulo 2 (Desenvolvimento) onde é apresentado e explicado o código que foi desenvolvido. No Capítulo 3 (Resultados) é demonstrado o funcionamento do programa e no Capítulo 4 apresentam-se as conclusões do projeto. Para acabar, é comentada a divisão do trabalho entre os elementos do grupo no Capítulo 5.

Capítulo 2

Desenvolvimento

2.1 Python

2.1.1 Módulos

Os módulos usados foram os seguintes:

- os.path: obtenção de informações sobre diretórios;
- cherrypy: usado para o desenvolvimento da aplicação;
- random e string: ambos usados para gerar os tokens, uma string de 16 caracteres ASCII aleatórios;
- sqlite3: usado para operações relacionadas com a base de dados (incluindo criação e métodos);
- json: funções de retorno de informação e para obtenção da mesma.

O programa começa com a definição de um dicionário que contribui para a configuração da aplicação (Figura 2.1).

2.1.2 Base de Dados

Como podemos verificar na figura abaixo (Figura 2.2), toda a função (*def create(sel, username, password):*) é usada para a criação da tabela, caso não

```
# Dictionary with this application's static directories configuration
config = {
    "/": {"tools.staticdir.root": baseDir},
    "/html": {"tools.staticdir.on": True,
              "tools.staticdir.dir": "html"},
    "/js": {"tools.staticdir.on": True,
            "tools.staticdir.dir": "js"},
    "/css": {"tools.staticdir.on": True,
             "tools.staticdir.dir": "css"},
    "/images": {"tools.staticdir.on": True,
                "tools.staticdir.dir": "images"},
    "/uploads": {"tools.staticdir.on": True,
                 "tools.staticdir.dir": "uploads"},

}
```

Figura 2.1: Dicionário que contribui para a configuração da aplicação

exista, será criada uma. A tabela guarda informações como: o nome de utilizador e a password.

```
    def create(self, username, password):
        # Create a new user in the database
        with sql.connect("database.db") as db:
            c = db.cursor()

            # If user doesn't exist, create it, else return False
            c.execute("SELECT * FROM users WHERE username=?", (username,))
            user = c.fetchone()
            if not user:
                c.execute("INSERT INTO users (username, password) VALUES (?, ?)",
                          (username, password))
                db.commit()
                return True
            else:
                return False
```

Figura 2.2: Criação da base de dados

Na Figura 2.3 a função *delete* remove um utilizador da base de dados e a função delete_token remove o token do utilizador.

a)

```python
    def delete(self, username):
        # Delete a user from the database
        with sql.connect("database.db") as db:
            c = db.cursor()

            # If user exists, delete it, else return False
            c.execute("SELECT * FROM users WHERE username=?", (username,))
            user = c.fetchone()
            if user:
                c.execute("DELETE FROM users WHERE username=?", (username,))
                db.commit()
                return True
            else:
                return False
```

b)

```python
    def delete_token(self, token):
        # Delete a user's token from the database
        with sql.connect("database.db") as db:
            c = db.cursor()
            # If token exists, delete it, else return False
            c.execute("SELECT * FROM users WHERE token=?", (token,))
            user = c.fetchone()
            if user:
                c.execute("UPDATE users SET token=NULL WHERE token=?", (token,))
                db.commit()
                return True
            else:
                return False
```

Figura 2.3: a) Remoção do utilizador da base de dados; b) Remoção do token de utilizador da base de dados.

Na Figura 2.4 a função *valid* "checa"se o token está na base de dados.

```python
    def valid(self, token):
        # Check if token is in database
        with sql.connect("database.db") as db:
            c = db.cursor()
            c.execute("SELECT * FROM users WHERE token=?", (token,))
            user = c.fetchone()
            if user:
                return True
            else:
                return False
```

Figura 2.4: Validação do token

Por fim a Figura 2.5 refere-se à função *auth* que, caso o utilizador e a password estejam corretas cria um token *American Standard Code for Information Interchange (ASCII).*

```python
    def auth(self, username, password):
        # If user and password are correct, create a ascii token, else return None
        with sql.connect("database.db") as db:
            c = db.cursor()
            c.execute("SELECT * FROM users WHERE username=? AND password=?",
                      (username, password))
            user = c.fetchone()

            if user:
                # Random ascii token with 16 characters
                token = ''.join(random.choices(
                    string.ascii_letters + string.digits, k=16))
                c.execute("UPDATE users SET token=? WHERE username=?",
                          (token, username))
                db.commit()
                return {
                    "authentication": "OK",
                    "token": token
                }

            else:
                return None
```

Figura 2.5: Criação do um token ASCII

2.1.3 Interligação das Páginas

Posteriormente passámos à interligação das páginas que constituíam a interface da aplicação.

```python
    # HTML pages
    @cherrypy.expose
    def index(self, error=None):
        return open("html/login.html")

    @cherrypy.expose
    def register(self, error=None):
        return open("html/register.html")

    @cherrypy.expose
    def home(self, token=None):
        # if token is in data base, return home page
        # else return login page

        if token is not None:
            if self.users.valid(token):
                return open("html/home.html")
        raise cherrypy.HTTPRedirect("/")

    @cherrypy.expose
    def about(self, token=None):
        return open("html/about.html")

    @cherrypy.expose
    def collections(self, token=None, id=None):
        # if token is in data base, return collections page
        # else return login page
        if token is not None:
            if self.users.valid(token):
                return open("html/collections.html")
        raise cherrypy.HTTPRedirect("/")

    @cherrypy.expose
    def create(self, token=None):
        # if token is in data base, return create page
        # else return login page

        if token is not None:
            if self.users.valid(token):
                return open("html/create.html")
        raise cherrypy.HTTPRedirect("/")
```

Figura 2.6: Interligação das páginas

2.1.4 Sign in, Sign out e Sign up

Nesta subseção veremos as funções *sign_in*, *sign_out* e *sign_up* que servem, respetivamente para fazer o login na página *login.html*, para fazer o logout e para fazer um registo novo de utilizador.

```
78      @cherrypy.expose
79      def sign_out(self, token=None):
80
81          if token is not None:
82              self.users.delete_token(token)
83          raise cherrypy.HTTPRedirect("/")
84
85      @cherrypy.expose
86      def sign_in(self, username=None, password=None):
87          if username is not None and password is not None:
88              reply = self.users.auth(username, password)
89              if reply is not None:
90                  raise cherrypy.HTTPRedirect("/home?token=" + reply["token"])
91          raise cherrypy.HTTPRedirect("/?error=1")
92
93      @cherrypy.expose
94      def sign_up(self, username='', password=''):
95          # if username and password are not empty, create user
96          # if username in data base, return error message
97          # else return login page
98          if username != '' and password != '':
99              if self.users.create(username, password):
100                 raise cherrypy.HTTPRedirect("/")
101         raise cherrypy.HTTPRedirect("/register?error=1")
```

Figura 2.7: Funções sign's

2.1.5 Coleções e uploads

A figura abaixo (Figura 2.8) representa duas funções, uma relativa à criação de uma nova coleção (*new_collection*) e outra ao upload das imagens de cada utilizador (*upload*).

```
107     @cherrypy.expose
108     def new_collection(self, token=None, collection_name=''):
109         # if collection_name is not empty
110         # if collection_name in data base, return error message
111         # else return login page
112
113         if collection_name != '':
114             # Check if collection_name is in database
115             if not self.cromos.collection_exists(collection_name):
116                 return json.dumps({"status": "OK"})
117         return json.dumps({"status": "false"})
118
119     @cherrypy.expose
120     def upload(self, myFile):
121         with open("uploads/" + myFile.filename, "wb") as fo:
122             while True:
123                 data = myFile.file.read(8192)
124                 if not data:
125                     break
126                 fo.write(data)
```

Figura 2.8: Funções criadoras de coleções e upload de imagens

2.1.6 Configuração e porta TCP

Finalizando esta seção temos na Figura 2.9 uma configuração da aplicação e da porta Transmission Control Protocol (TCP) usada.

```
23   # Configure socket port
24   cherrypy.config.update({'server.socket_port': 10010, })
```

Figura 2.9: Configuração da porta

2.2 HTML, CSS e JavaScript

A interface Web foi implementada para uma versão desktop. Os ficheiros *.html* das mesmas podem ser facilmente identificados na pasta *html* no diretório principal. A versão é composta por seis ficheiros do tipo HyperText Markup Language (HTML), nomeadamente: *login.html*, *register.html*, *home.html*, *collection.html*, *create.html* e *about.html*. Cada um destes ficheiros representa uma página com um propósito específico de interação com o utilizador.

Vejamos a implementação da página *home.html* e serão comentadas as seguintes páginas apenas nas suas variações.

O head da página *home.html* é "parecido"ao das restantes. Aqui se apresenta o seu código:

```
1   <!DOCTYPE html>
2   <html>
3
4   <head>
5       <title>Caderneta de Cromos - Início</title>
6       <meta charset="UTF-8">
7       <meta name="viewport" content="width=device-width, initial-scale=1.0">
8       <meta http-equiv="X-UA-Compatible" content="ie=edge">
9       <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
10      <script src="../js/functions.js"></script>
11      <link rel="stylesheet" href="../css/navBarStyle.css">
12      <link rel="stylesheet" href="../css/homeStyle.css">
13  </head>
```

Figura 2.10: HEAD da página *home.html*

Tal como se pode observar, as primeiras linhas destinam-se a definir os dados *meta*, que poderão ser usados como por exemplo por motores de busca. Define-se também o título, os *links* para os ficheiros Cascading Style Sheets (CSS) e os *script's* (JavaScript).

Na construção do website foi implementado uma barra de navegação no topo da página que permite acesso ao resto do conteúdo. O bloco de navegação:

É de notar que o que muda em todas as páginas HTML é precisamente o

```
16      <nav>
17          <ul class="menu">
18              <li class="logo"><a href="https://www.ua.pt/"><img src="../images/uaLogo.png" width="50px"
19                          height="50px"></a></li>
20              <li class="item"><a href="#" id="home">Início</a></li>
21              <li class="item"><a href="#" id="collections">Coleções</a></li>
22              <li class="item"><a href="#" id="about">Sobre</a></li>
23              <li class="item button"><a href="#" id="signout">Sair</a></li>
24              <li class="toggle"><span class="bars"></span></li>
25          </ul>
26      </nav>
```

Figura 2.11: Bloco de navegação

conteúdo da Figura 2.10 e os blocos com os conteúdos das páginas restantes, isto visto o esquema de navegação ser igual em todas exceptuando as páginas *login.html* e *register.html*.

A página *login.html*, que se apresenta como página inicial, contém um *head* bastante semelhante ao referido anteriormente, diferindo em pequenos pormenores como os ficheiros CSS e ficheiros que contém os *Scripts* (ou seja, código *JavaScript*) e não apresenta barra de navegação. O mesmo se aplica à página *register.hmtl*.

Na página *create.html* o seu conteúdo encontra-se a cargo do JavaScript, sendo que este recebe as imagens inseridas pelo utilizador.

A página *collections.html* tem como objetivo disponibilizar as imagens relativas à adição feita na página anteriormente referida.

Por último, a página *about.hmtl* disponibiliza ao utilizador informações como os desenvolvedores da aplicação, um mapa com a identificação do DETI, a possibilidade de contactar cada um por e-mail e um pequeno texto informativo.

Capítulo 3

Resultados

Na Figura 3.1 podemos ver uma apresentação da página inicial da aplicação web desenvolvida, onde se pode fazer login ou então registo de utilizador.

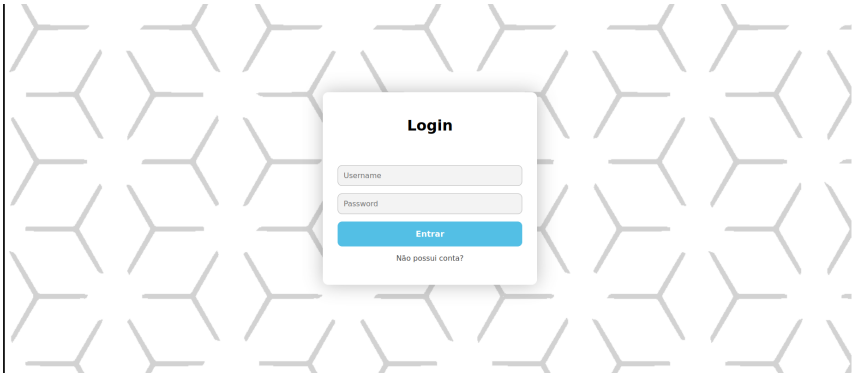

Figura 3.1: Página inicial

Se selecionarmos "Não possui conta?"abrirá a página relativa ao registo de novos utilizadores e se posteriormente fizermos *login* passaremos à página principal (*home.html*). Já nesta página o utilizador poderá ver as coleções (ou explorá-las por nome) ou então criar uma coleção, o que o levará à página *create.html*. A partir desta página é possível enviar uma imagem nova, sendo para esse efeito apenas necessário carregar no botão "Explorar..."e escolher a imagem a inserir na coleção.

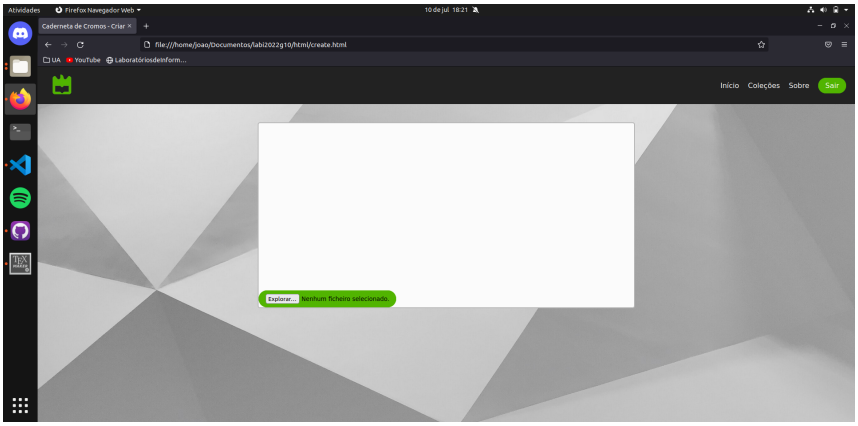

Figura 3.2: Página *create*

Após isso a imagem é apresentada e enviada automaticamente para uma pasta, *uploads*, onde ficará guardada. Quando a coleção estiver de acordo com o pretendido do utilizador, o mesmo poderá, na aba de navegação, percorrer até à página *Coleções* e ver as imagens enviadas.

3.1 Objetivos não cumpridos

Não foi possível implementar todas as funcionalidades pedidas tal como o processador de imagens cujo objetivo seria lidar com as imagens enviadas para o sistema e a obtenção das mesmas. Em particular deveria suportar a obtenção de uma ou mais imagens. Também não foi possível usar as cifras e colocar marcas de água nas imagens. Estas "falhas"ocorreram devido à má gestão de tempo.

Capítulo 4

Conclusão

Neste projeto foi possível consolidar e melhorar os conhecimentos adquiridos ao longo das aulas deste segundo semestre (SQLite3, JSON, CherryPy), como também rever os conhecimentos do semestre anterior (HTML, CSS e JS), contribuindo assim para a valorização do nosso percurso académico.

Com este projeto conseguimos adquirir competências nas áreas de desenvolvimento de aplicações web, manipulação de imagem e bases de dados, para além das competências de aprendizagem que conseguimos atingir, também desenvolvemos de forma proativa o trabalho em equipa (discussão e aceitação de ideias). Apesar das possíveis dificuldades que enfrentamos, procurámos resolver os nossos problemas de forma autónoma e de forma interdependente dentro do nosso grupo de trabalho. Em trabalhos futuros consideramos importante a melhor gestão do tempo em relação a outras UC's, reconhecemos no entanto que trabalhámos arduamente para o que apresentámos aqui.

Concluímos que foi uma experiência que nos enriqueceu a vários níveis para um futuro mercado de trabalho, atualmente para o nosso percurso académico e enquanto pessoas.

Contribuições dos autores

A divisão do trabalho foi feita da seguinte forma:

Pedro Ferreira (PF) desenvolveu o python e a base de dados presente neste projeto.

Guilherme Santos (GS), Beatriz Oliveira (BO) e João Gaspar (JG) assumiram responsabilidade pelo design e aspecto gráfico da aplicação, tendo discutido e planeado a sua parte em conjunto.

É importante referir que apesar desta divisão todo o projecto foi discutido, planeado e estruturado pelos 4 membros. As percentagens representativas do trabalho de cada autor são:

- PF - 35%
- JG - 21%
- GS - 23%
- BO - 21%

O relatório foi escrito pelo JG e pela BO.

Foi usado um repositório na plataforma CodeUA para o qual todos os autores contribuiu com commits frequentes. O link para esse repositório encontra-se na página *about.html* da aplicação desenvolvida.

Acrónimos

JSON JavaScript Object Notation

HTML HyperText Markup Language

ASCII American Standard Code for Information Interchange

TCP Transmission Control Protocol

CSS Cascading Style Sheets

JS JavaScript

UC Unidade Curricular

PF Pedro Ferreira

JG João Gaspar

GS Guilherme Santos

BO Beatriz Oliveira