REPÚBLICA PORTUGUESA
EDUCAÇÃO

AEPA
Agrupamento de Escolas Pedro Alexandrino
Póvoa de Santo Adrião

ESCOLA SECUNDÁRIO PEDRO ALEXANDRINO

Prova de Aptidão Pedagógica

CURSO: Técnico de Equipamentos Informáticos

TURMA: 3TEI CÓDIGO SIGO: 8162722

Realizado por: Francisco Emanuel Ferreira Fadigas

Ano Letivo: 2020/21

Índice

Conteúdos

Introdução

Este projeto, desenvolvido no âmbito da Prova de Aptidão Profissional do Curso Técnico de Gestão de Equipamentos Informáticos, visa englobar todos os conhecimentos adquiridos ao longo deste ciclo de estudos. Tem como nome NxGen File Manager, e consiste num dispositivo cujo objetivo é automatizar a transferência de ficheiros entre dois suportes digitais diferentes.

O projeto tem como objetivo ajudar o utilizador, projetado como um fotógrafo de rua, que precise rapidamente de transferir ficheiros de um cartão SD para um disco externo, mas graças à adaptabilidade do dispositivo, pode ser também usado para outros fins, como a transferência entre dois discos externos, flash drives USB, ou o que o utilizador preferir.

Desenvolvimento

A ideia para este projeto surgiu após eu me ter deparado na exata situação para o qual esta PAP foi desenvolvida: Encontrava-me na rua a tirar fotos quando fiquei sem espaço no cartão SD, e fiquei a pensar como é que poderia resolver esse problema, e encontrei a solução num microcomputador onde pudesse ligar tanto o cartão SD como um disco externo.

O unico microcomputador que preencheu esses requisitos de forma relativamente económica foi o *Raspberry Pi 4*, equipado com duas portas USB 3.1 Gen 1x1, que garantem velocidades rápidas e estáveis, bem como duas portas USB 2.0, uma micro HDMI e uma saida de som 3.5'' caso se queira usar o dispositivo como computador ultra-portátil.

O *Raspberry Pi* é uma série de microcomputadores introduzida no mercado em 2012, que se revelou a escolha perfeita para ser a peça central deste projeto por várias razões. Primeiro, é bastante pequeno, revelando-se bastante adaptável para qualquer cenário, quer seja para ficar lado a lado com um computador, seja ele laptop ou desktop, ou vá o destino dele ser ficar guardado numa mochila para quando for necessário.

Adicionalmente, é um dispositivo bastante potente, dando uso a um microprocessador *BMC2711*, que usa o conjunto de instruções *ARMV8*, sendo capaz de dar energia a um circuito personalizado através dos pinos *GPIO* (*General Purpose Input/Output*), standard presente em quase todos os modelos deste microcomputador.

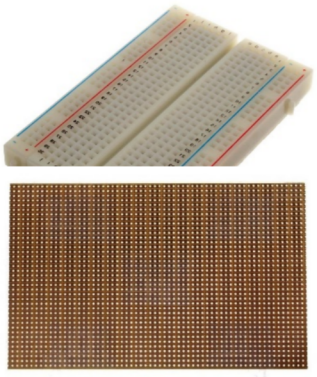
Materiais utlizados

Os materiais usados foram os seguintes:

Descrição:	Raspberry PI 4 Model B 4GB	
Características	CPU: Broadcom BCM2711 (ARMV8) 64-bit @ 1.5GHz Memória: 4GB LPDDR4-3200MHz SDRAM Conectividade Wireless: Wi-Fi Dual-Band 802.11ac, Bluetooth 5.0, BLE Ethernet: 10/100/1000 Mbps Conectividade USB: 2 portas USB 3.2 Gen 1x1, 2 Portas USB 2.0 Saída Gráfica: 2 portas Micro HDMI (até 4kp60) Saída Áudio: Áudio Estéreo de 4 polos (Também serve de saída de vídeo composto) Armazenamento: Entrada Micro-SD	 

Descrição:	Cabo HDMI «» micro-HDMI m/m - 1.5m	
Características	Usado para obter saída de vídeo para testagem e *debug* dos scripts *Python*	

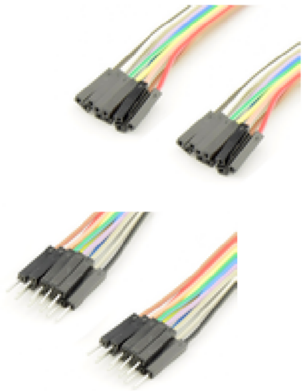
Descrição:	Fonte de alimentação USB-C (230VAC->5.1VDC) 3.0A 15.3W - preto - oficial para *Raspberry Pi 4*	
Características	Usada para alimentar o *Raspberry Pi* enquanto era usado em ambientes estáticos	

Descrição:	Breadboard Genérica	
Características	Usada na testagem dos componentes que interagem com o *Raspberry PI* através dos pinos GPIO	

Descrição:	Placa de circuito perfurada 3 pontos 160x100mm	
Características	Usada da mesma forma que a Breadboard, mas para uso em movimento	

Descrição:	Resistência de filme metálico 820R 0.6W ±1% Ø2.5x6.8mm	
Características	Regula a corrente que atinge os LEDs	

Descrição:	LED 1.8mm com base quadrada vermelho 8-15mcd 70º - Kingbright L-2060ID && LED 1.8mm com base quadrada verde 5-10mcd 70º - Kingbright L-2060GD	
Características	Usados para notificar o utilizador do estado do dispostivo	

Descrição:	Conjunto de 10 cabos de ligação Jumper Dupont macho-macho - 150mm && Conjunto de 10 cabos de ligação Jumper Dupont fêmea-fêmea - 150mm	
Características	Usado para realizar a comunicação entre a breadboard e os LEDs NOTA: À data de aquisição do material, não se encontravam disponíveis cabos macho-fêmea, pelo que tive de adquirir os dois em separado	

Descrição:	Botão miniatura 12x12mm SPST-NO 12VDC 50mA THT 1.6N	
Características	Usado pelo utilizador para indicar ao Raspberry pi que os dispositivos de entrada e saída estão ligados	

Descrição:	Powerbank Xiaomi Redmi 2 20000mAh 18W Fast Charge Branca	
Características	Usado para alimentar o Raspberry Pi em ambientes móveis	

Para funcionamento do projeto, desta lista de material são apenas necessários o *Raspberry Pi 4*, os *LEDs* e a resistência, a Breadboard ou a Placa Perfurada e a fonte de alimentação ou o *Powerbank*, dependendo do uso que se vai dar ao equipamento (local ou móvel). O *Powerbank* apresentado é meramente ilustrativo e a escolha deste modelo específico é opcional, visto que o *Raspberry Pi* é alimentado por qualquer fonte de alimentação genérica (desde que seja capaz de alimentar o circuito a 5v 3A).

Adicionalmente, tinha sido projetada a criação de uma caixa em 3d para melhor utilização do dispositivo em movimento, contudo devido à pandemia do COVID-19, a sua impressão foi atrasada, mas visualmente será algo semelhante às imagens abaixo:

Para a configuração inicial é, contudo, necessário que o utilizador corra alguns scripts de instalação, pelo que é recomendado, caso o utilizador não possua já, a aquisição de um cabo micro HDMI para HDMI, um teclado e um rato, ambos USB, para a realização da configuração.

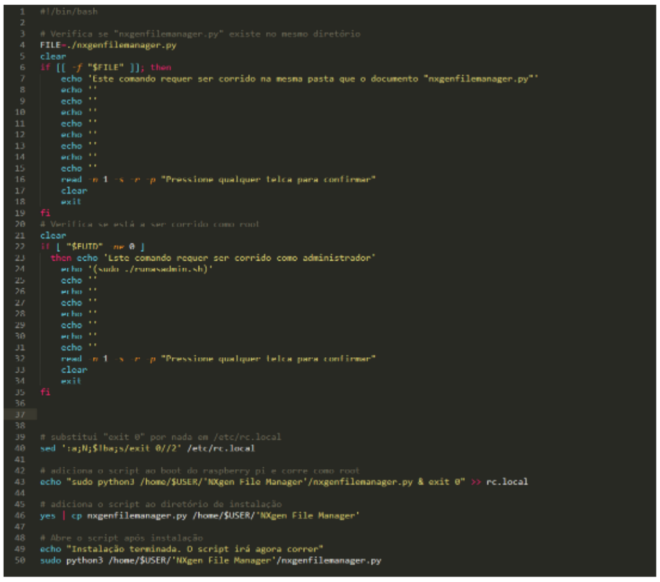
Procedimentos

Este projeto é composto de dois scripts, um que vai correr permanentemente no background (semelhante a um daemon), e outro que serve para instalar o primeiro.

O utilizador deve correr primeiro o script de instalação ("runasadmin.sh") que, como o nome indica, deve ser corrido como **admin**istrador (através do comando *sudo ./runasadmin.sh*)

Este script funciona de forma simples e foi escrito em *Bash*, de forma a interagir diretamente com o *kernel* Linux do *Raspberry Pi*, que corre uma distribuição personalizada, chamada Rasperry Pi OS, baseada em Debian, outra distribuição Linux muito conhecida (outras distribuições Linux que tomam Debian como base são Ubuntu, Kali Linux, Linux Mint, Pop! OS, elementary OS, Tails, entre inúmeras outras)

O script é o seguinte:



```
#!/bin/bash

# Verifica se "nxgenfilemanager.py" existe no mesmo diretório
FILE=./nxgenfilemanager.py
clear
if [[ -f "$FILE" ]]; then
    echo 'Este comando requer ser corrido na mesma pasta que o documento "nxgenfilemanager.py"'
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    read -n 1 -s -r -p "Pressione qualquer tecla para confirmar"
    clear
    exit
fi
# Verifica se está a ser corrido como root
clear
if [ "${EUID}" -ne 0 ]
  then echo 'Este comando requer ser corrido como administrador'
    echo '(sudo ./runasadmin.sh)'
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    read -n 1 -s -r -p "Pressione qualquer tecla para confirmar"
    clear
    exit
fi


# substitui "exit 0" por nada em /etc/rc.local
sed '1;a;N;$!ba;s/exit 0//2' /etc/rc.local

# adiciona o script ao boot do raspberry pi e corre como root
echo "sudo python3 /home/$USER/'NXgen File Manager'/nxgenfilemanager.py & exit 0" >> rc.local

# adiciona o script ao diretório de instalação
yes | cp nxgenfilemanager.py /home/$USER/'NXgen File Manager'

# Abre o script após instalação
echo "Instalação terminada. O script irá agora correr"
sudo python3 /home/$USER/'NXgen File Manager'/nxgenfilemanager.py
```

Vamos decompô-lo e explicá-lo por cada comentário:

```
1   #!/bin/bash
```

Esta primeira linha tem o nome de "*shebang*". Serve para dizer à shell *BASH* para executar os comandos abaixo. Tem a mesma funcionalidade que o *<!DOCTYPE html>* no html

```
# Verifica se "nxgenfilemanager.py" existe no mesmo diretório
FILE=./nxgenfilemanager.py
clear
if [[ -f "$FILE" ]]; then
    echo 'Este comando requer ser corrido na mesma pasta que o documento "nxgenfilemanager.py"'
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    read -n 1 -s -r -p "Pressione qualquer telca para confirmar"
    clear
    exit
fi
```

Tal como o comentário diz, esta rotina cria uma variável, *FILE*, com o nome (*nxgenfilemanager.py*) e diretório (*./*, ou seja, o diretório atual, de onde corre o script) esperado do instalador. Logo de seguida, limpa o ecrã para melhorar a visibilidade. Se não encontrar o ficheiro designado à variável, mostra ao cliente uma mensagem de aviso e aguarda que ele prima uma tecla qualquer. Logo de seguida, volta a limpar o ecrã e fecha o *script*

```
# Verifica se está a ser corrido como root
clear
if [ "$EUID" -ne 0 ]
  then echo 'Este comando requer ser corrido como administrador'
    echo '(sudo ./runasadmin.sh)'
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    echo ''
    read -n 1 -s -r -p "Pressione qualquer telca para confirmar"
    clear
    exit
fi
```

Este segmento é semelhante ao anterior, verificando se o *script* está a ser corrido como administrador, e avisando o utilizador caso se verifique o contrário.

```
# substitui "exit 0" por nada em /etc/rc.local
sed ':a;N;$!ba;s/exit 0//2' /etc/rc.local
```

Em Shells *BASH*, o comando *sed* serve para trocar *strings* por outras em ficheiros, com a sintaxe *sed '[regras regex]/[string a encontrar]/[string a escrever]/[número da string a sobrescrever]' [ficheiro].* Neste caso, o comando procura a segunda *string* "*exit 0*" (a primeira é um comentário a dizer para não se remover esta *string*), e remove-a (mas não se assustem, será adicionada de volta posteriormente). O objetivo disto é preparar o ficheiro *rc.local* (situado em */etc/*) para ser editado. Este ficheiro corre os scripts que estejam lá explicitados como *root* (utilizador administrador no Linux), assim que o dispositivo se liga.

```
# adiciona o script ao boot do raspberry pi e corre como root
echo "sudo python3 /home/$USER/'NXgen File Manager'/nxgenfilemanager.py & exit 0" >> rc.local
```

Esta linha utiliza o comando *echo* para escrever o comando que o Raspberry deve correr para iniciar o script, com o caracter *&*, e volta a adicionar a linha "*exit 0*" ao ficheiro rc.local. Esta string é essencial, pois caso ela não exista o dispositivo não permite dar login. O caracter *&* é adicionado para que o script corra em segundo plano e o Raspberry continue com o *boot* e passe para o *login*.

```
45  # adiciona o script ao diretório de instalação
46  yes | cp nxgenfilemanager.py /home/$USER/'NXgen File Manager'
```

Esta linha "empurra" a palavra "*yes*" para o comando *cp*, usado para copiar um ficheiro para um diretório. Isto serve para que, caso o ficheiro já exista, a confirmação para sobrescrever o ficheiro seja automática. Os mais observadores são capazes de reparar que "*$USER*" não é um nome usual para um utilizador em qualquer sistema operativo. Isto deve-se ao facto que *$USER* é, na realidade, uma *environment variable* (variável de ambiente) que dá *return* ao utilizador que corre o script. Isto garante ao instalador uma maior versatilidade, adaptando-se a qualquer nome, caso o utilizador o tenha mudado.

```
48  # Abre o script após instalação
49  echo "Instalação terminada. O script irá agora correr"
50  sudo python3 /home/$USER/'NXgen File Manager'/nxgenfilemanager.py
```

Esta linha muito simplesmente corre o *script*, agora do diretório instalado. Isto serve tanto para dar início ao *setup*, como para confirmar se o instalador correu como espectável.

Passando para o script principal, este assume a seguinte estrutura:

Novamente, vamos reduzir isto em excertos mais legíveis.

O script começa com estas duas linhas:

```
1  import os
2  os.system('clear')
```

O que estas linhas fazem é importar a livraria "*os*", que serve para comunicar com o resto do sistema operativo, e limpa a consola na segunda linha, lançando o comando "*clear*" para o terminal.

A próxima linha que corre neste script é a função *main()*.

```
226 def main():
227     mod_check()
228     setup()
229     while 1 > 0:
230         import RPi.GPIO as GPIO
231         import time
232         GPIO.setmode(GPIO.BCM)
233         GPIO.setup(7, GPIO.OUT) # LED 1: Ligado
234         GPIO.setup(8, GPIO.OUT) # LED 2: A trabalhar
235         GPIO.setup(10, GPIO.OUT) # LED 3: Sucesso
236         GPIO.setup(11, GPIO.OUT) # LED 4: Erro
237         GPIO.output(7, False)
238         GPIO.output(8, False)
239         GPIO.output(10, False)
240         GPIO.output(11, False)
241
242         exif_read()
243 main()
```

A função *main()*, tanto é corrida como é definida quase no fim do script. Isto faz com que as outras funções sejam definidas e lidas para a memória, de modo a evitar erros. No fundo, esta função serve como "casa" do programa, em que sequencialmente são corridos os vários passos para pôr a correr o programa. Podemos, assim, ver que a primeira "sub-função" que é corrida é a *mod_check()*. Esta tem como objetivo assegurar que os módulos necessários para correr este programa estão instalados. O ciclo *while* será explicado mais adiante, ao mesmo tempo que *exif_read()*.

```python
# Procura pelos módulos necessários, instala se não existirem, reinicia até instalar corretamente
def mod_check():
    import os
    import time
    import colorama
    from colorama import Fore, Back, Style

    print('Verificando a existência de módulos necessários')
    check_mod = os.popen("pip3 list| grep ExifRead").read()
    if check_mod[0:8] == "ExifRead":
        print('Módulo "ExifRead" presente, iniciando')
    else:
        os.system("pip3 install ExifRead")
        install_mod = os.popen("pip3 list| grep ExifRead").read()
        if install_mod[0:8] == "ExifRead":
            print('Módulo "ExifRead" instalado com sucesso')
        else:
            print('Módulo "ExifRead" não instalado, reiniciando')
            mod_check()
    check_mod = os.popen("pip3 list| grep colorama").read()
    if check_mod[0:8] == "colorama":
        print('Módulo "colorama" presente, iniciando')
    else:
        os.system("pip3 install colorama")
        install_mod = os.popen("pip3 list| grep colorama").read()
        if install_mod[0:8] == "colorama":
            print('Módulo "colorama" instalado com sucesso')
        else:
            print('Módulo "colorama" não instalado, reiniciando')
            mod_check()
    check_mod = os.popen("pip3 list| grep configparser").read()
    if check_mod[0:12] == "configparser":
        print('Módulo "configparser" presente, iniciando')
    else:
        os.system("pip3 install configparser")
        install_mod = os.popen("pip3 list| grep configparser").read()
        if install_mod[0:12] == "configparser":
            print('Módulo "configparser" instalado com sucesso')
        else:
            print('Módulo "configparser" não instalado, reiniciando')
            mod_check()
    check_mod = os.popen("pip3 list| grep pathlib").read()
    if check_mod[0:7] == "pathlib":
        print('Módulo "pathlib" presente, iniciando')
    else:
        os.system("pip3 install pathlib")
        install_mod = os.popen("pip3 list| grep pathlib").read()
        if install_mod[0:7] == "pathlib":
            print('Módulo "pathlib" instalado com sucesso')
        else:
            print('Módulo "pathlib" não instalado, reiniciando')
            mod_check()
    time.sleep(3)

```

A função *mod_check()* é mais comprida, contudo tem uma funcionalidade muito simples e repetitiva. Essencialmente, aproveita-se também da livraria "*os*" para correr comandos no terminal. Neste caso, o comando é *pip3 list | grep [módulo a verificar]*. O que este comando faz é que lista todos os módulos instalados, e verifica se na lista se encontra o módulo que se pretende verificar. Caso o comando conclua com sucesso, exibe a mensagem *"Módulo [nome do módulo] presente, iniciando"*. Caso contrário, tenta instalar o módulo em falta, e se concluir esse objetivo, exibe a mensagem *"Módulo [nome do módulo] instalado com sucesso"*, caso contrário, diz que o módulo não foi instalado e reinicia o processo até conseguir. Após o fim deste processo, espera 3 segundos para o utilizador poder confirmar que o último módulo passou o teste com sucesso.

```
142 def setup():
143
144     import configparser
145     from pathlib import Path
146     from os import path
147     USER = "/" + os.popen("echo $USER").read() + "/"
148     os.system('clear')
149     config = configparser.ConfigParser()
150     if path.exists("/home" + USER + "NXgen File Manager/settings.ini") == True:
151         config.read("/home" + USER + "NXgen File Manager/settings.ini")
152         if config.has_option('Directories','input') == True:
153             if config.get('Directories','input') == "yes" or config.get('Directories','input') == "":
154                 setup_input()
155         if config.has_option('Directories','output') == True:
156             if config.get('Directories','output') == "yes" or config.get('Directories','output') == "":
157                 setup_output()
158         if config.has_option('Directories','struct') == True:
159             if config.get('Directories','struct') == "yes" or config.get('Directories','struct') == "":
160                 struct_output()
161         else:
162             exif_read()
163     elif path.exists("/home" + USER + "NXgen File Manager/settings.ini") == False:
164         init_def()
```

A próxima função a correr é a *setup()*.

O que esta função faz é garantir que o ficheiro das definições se encontra povoado com os dados necessários. Isto deve-se a uma peculiaridade na livraria escolhida para ler os ficheiros de configuração, que necessita que estes se encontrem pré-povoados antes de serem lidos.

Caso o ficheiro exista, é verificado se os valores lá inseridos diferem dos valores pré definidos (de modo a nunca correr o *setup* quando for desnecessário), e caso contrário, corre a função *init_def()*, que será explicitada abaixo.

```
59 def init_def():
60
61     #Inicializa as definições caso não existam com valores predefinidos, futuramente sobreescritos
62     from pathlib import Path
63     from os import path
64     import configparser
65     USER = os.popen("echo $USER").read()
66     if not os.path.exists("/home" + USER + "NXgen File Manager/"):
67             os.makedirs("/home" + USER + "NXgen File Manager/")
68     config = configparser.ConfigParser()
69     if not os.path.exists("/home" + USER + "NXgen File Manager/settings.ini"):
70         Path("/home" + USER + "NXgen File Manager/settings.ini").touch()
71     config.read("/home" + USER + "NXgen File Manager/settings.ini")
72     if config.has_section("Directories") == False:
73         config.add_section("Directories")
74     if config.has_option("Directories","input") == False:
75         config.set("Directories","input","yes")
76         with open("/home" + USER + "NXgen File Manager/settings.ini","w") as settingsfile:
77                 config.write(settingsfile)
78     if config.has_option("Directories","output") == False:
79         config.set("Directories","output","yes")
80         with open("/home" + USER + "NXgen File Manager/settings.ini","w") as settingsfile:
81                 config.write(settingsfile)
82     if config.has_option("Directories","struct") == False:
83         config.set("Directories","struct","yes")
84         with open("/home" + USER + "NXgen File Manager/settings.ini","w") as settingsfile:
85                 config.write(settingsfile)
86     setup()
87
```

O que esta função faz é definir uma variável (*USER*) como o nome do utilizador que corre o ficheiro. Posteriormente, verifica se o diretório onde se encontra o ficheiro das definições existe, e caso não exista, cria-o. Logo de seguida, verifica se o ficheiro de configuração existe, e caso contrário, cria-o.

Imediatamente após criar ou encontrar o ficheiro de configuração, povoa as definições de diretório de entrada e saída, bem como se a estrutura da *output* foi definida com o valor *"yes"*, escolhido arbitrariamente para ocupar momentaneamente o ficheiro de definições.

Este ficheiro, após estar escrito, terá a seguinte aparência:

```
[Directories]
input = yes
output = yes
struct = yes
```

Após a conclusão de todos estes *checks*, corre a função *setup()* novamente.

Observá-la-emos novamente, desta vez com os ficheiros preenchidos em mente

```
def setup():

    import configparser
    from pathlib import Path
    from os import path
    USER = "/" + os.popen("echo $USER").read() + "/"
    os.system('clear')
    config = configparser.ConfigParser()
    if path.exists("/home" + USER + "NXgen File Manager/settings.ini") == True:
        config.read("/home" + USER + "NXgen File Manager/settings.ini")
        if config.has_option('Directories','input') == True:
            if config.get('Directories','input') == "yes" or config.get('Directories','input') == "":
                setup_input()
        if config.has_option('Directories','output') == True:
            if config.get('Directories','output') == "yes" or config.get('Directories','output') == "":
                setup_output()
        if config.has_option('Directories','struct') == True:
            if config.get('Directories','struct') == "yes" or config.get('Directories','struct') == "":
                struct_output()
        else:
            exif_read()
    elif path.exists("/home" + USER + "NXgen File Manager/settings.ini") == False:
        init_def()
```

Agora que os ficheiros estão povoados, podemos verificar que a função procura por *strings* na região do ficheiro de configuração referentes à *input*, *output* e "*struct*" (definida avante) que sejam "*yes*" ou nulas.

As funções iniciadas como resultado desses *checks* são *setup_input()*, *setup_output()* e *struct_output()*. As funções *setup_input()* e *setup_output()* são bastante semelhantes, pelo que exemplificarei apenas uma delas

```
def setup_input():

    import configparser
    config = configparser.ConfigParser()
    from colorama import Fore, Back, Style
    USER = os.popen("echo $USER").read()
    config.read("/home" + USER + "NXgen File Manager/settings.ini")
    print(Fore.RED + "Nos próximos passos, irá personalizar as suas definições")
    print(Style.RESET_ALL + "Que diretório define como input? (geralmente será o cartão SD)")
    print("Escreva apenas o nome do diretório, como por exemplo, se os diretórios listados forem")
    print(Fore.GREEN + "["Pasta1", "Pasta2"]," + Fore.BLACK + "escreva apenas \"Pasta1\" (sem aspas)")
    print(" ")
    print(" ")
    print(Style.RESET_ALL + "Vamos listar os diretórios disponíveis" + Fore.GREEN)
    print(os.listdir("/media/" + USER))
    input_dir = input()
    Style.RESET_ALL
    config.set("Directories","input", input_dir)
    with open("/home" + USER + "NXgen File Manager/settings.ini","w") as settingsfile:
        config.write(settingsfile)
```

A função *setup_input* adiciona um novo módulo, *"colorama"*. Este serve para mudar a cor do texto no terminal, de modo a aumentar a visibilidade de elementos-chave e melhorar a experiência do utilizador.

Este *script* informa o utilizador que deve escolher o seu diretório de *input*, e lista os disponíveis. O programa sabe que dispositivos estão montados pois acede à pasta */media/* respetiva do utilizador, diretório pré-definido de montagem de dispositivos de armazenamento de qualquer distribuição Debian.

```
def struct_output():

    import configparser
    from colorama import Fore, Back, Style
    USER = os.popen("echo $USER").read()
    config = configparser.ConfigParser()
    print(Fore.RED + "Agora, defina a estrutura de pastas para onde quer guardar os ficheiros. Pode usar qualquer tag de especificação EXIF usada pela sua câmara (Ex: "Image Make/Image Model/EXIF LensModel/", com aspas, com barra inicial, com barra final)")
    config.read("/home" + USER + "NXgen File Manager/settings.ini")
    struct_output = input()
    config.set("Directories","struct", struct_output)
    with open("/home" + USER + "NXgen File Manager/settings.ini","w") as settingsfile:
        config.write(settingsfile)
```

A função *struct_output()* requer que o utilizador esteja familiarizado com a forma como a sua câmara manipula a informação *EXIF* do processador. *EXIF* é um standard de meta dados de imagens, que guarda informação extra sobre as imagens, desde coisas triviais, como a data, até ao segundo, em que a fotografia foi tirada, até coisas mais técnicas, como o modelo da câmara, lente, etc., guardando até informação sobre onde foi tirada a foto, caso a câmara possua GPS. Devido à imensa variedade de fabricantes de câmaras, e até de ficheiros que cada fabricante aplica, era humanamente impossível programar cada opção para cada fabricante, pelo que deve ser o utilizador a encontrar as definições *EXIF* específicas da sua câmara. Por exemplo, eu possuo uma câmara Canon EOS 200D, e a minha configuração ficou como se segue:

```
1  [Directories]
2  input = EOS_DIGITAL
3  output = UNTITLED
4  struct = Image Make/Image Model/EXIF LensModel/
```

Essencialmente, a linha 4 (*struct*), define que a imagem será guardada tendo em conta a marca da Câmara (*Image Make*), o modelo da câmara (*Image Model*) e o tipo de lente (*EXIF Lens Model*).

Finalmente, o script passa para a função *exif_read()*, onde será lido o documento e copiados os ficheiros de *input* (linha 2) para *output* (linha 3) tendo em conta *struct* (linha 4). Observemos o código correspondente:

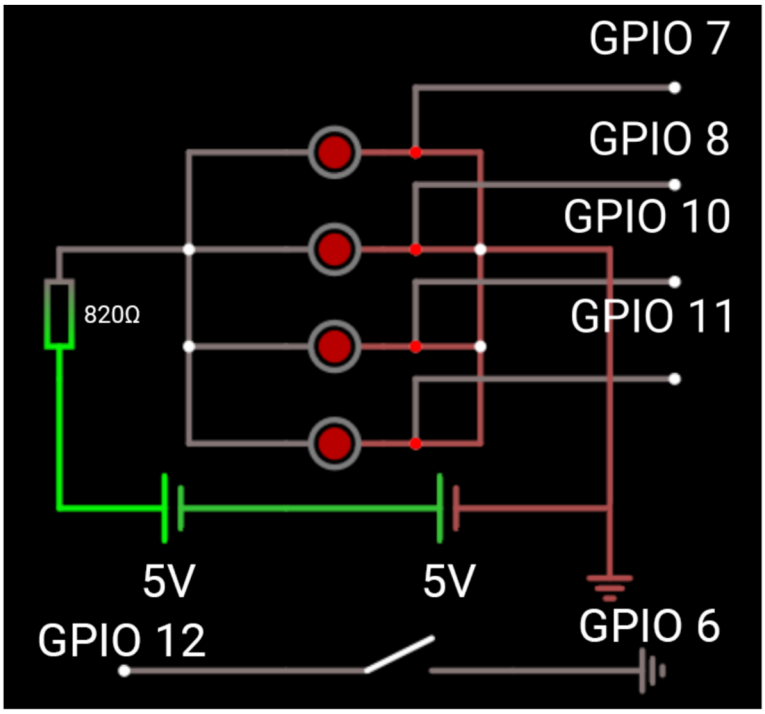
```
167 def exif_read():
168
169     import shutil
170     from shutil import copyfile
171     import configparser
172     from colorama import Fore, Back, Style
173     import exifread
174     import RPi.GPIO as GPIO
175     import time
176     GPIO.setmode(GPIO.BCM)
177     GPIO.setup(7, GPIO.OUT) # LED 1: Ligado
178     GPIO.setup(8, GPIO.OUT) # LED 2: A trabalhar
179     GPIO.setup(10, GPIO.OUT) # LED 3: Sucesso
180     GPIO.setup(11, GPIO.OUT) # LED 4: Erro
181     GPIO.setup(12, GPIO.IN) # BOTÃO
182     USER = os.popen("echo $USER").read()
183     valid_ext = (".tif", ".tiff", ".jpg", ".jpeg", ".cr2")
184     config = configparser.ConfigParser()
185     config.read("/home" + USER + "NXgen File Manager/settings.ini")
186     if config.has_option("Directories","input") == True:
187         inputdir = str("/media" + USER + config.get("Directories","input") + "/")
188         for root,subdir,files in os.walk(inputdir):
189             for curfile in files:
190                 curpath = os.path.join(root,curfile)
191                 if curpath.lower().endswith(valid_ext):
192                     with open(curpath, "rb") as image_file:
193                         tags = exifread.process_file(image_file)
194                     dest_dir = "/media" + USER + config.get("Directories", "output") + "/" + config.get("Directories", "struct") + curfile
195                     for tag in tags.keys():
196                         if tag not in ("JPEGThumbnail", "TIFFThumbnail", "Filename", "EXIF MakerNote"):
197                             key = str(tag)
198                             value = str(tags[tag])
199                             if "/" in value:
200                                 value = value.replace("/","_")
201                             #print("key: " + key + "value: " + value)
202                             dest_dir = dest_dir.replace(key, value)
203                     print(Fore.GREEN + dest_dir + Fore.WHITE)
204                     os.makedirs(os.path.dirname(dest_dir), exist_ok=True)
205                     shutil.copyfile(os.path.join(root,curfile), dest_dir)
206                     if not os.path.exists(dest_dir + "/" + os.path.join(root,curfile)):
207                             while GPIO.input(3) == GPIO.LOW:
208                                 GPIO.output(11, True)
209                                 time.sleep(0.016)
210                                 GPIO.output(11, False)
211                                 time.sleep(0.016)
212                                 GPIO.output(11, True)
213                                 time.sleep(0.016)
214                                 GPIO.output(11, False)
215                                 time.sleep(0.016)
216                                 GPIO.output(11, True)
217                                 time.sleep(0.016)
218                                 GPIO.output(11, False)
219                                 time.sleep.(0.016)
220                             else:
221                                 return
222
223             #if os.path.isdir(curfile) == True:
224                 #continue
225
226
```

Esta função, novamente define *USER* como o nome do utilizador, mas desta vez dá uso ao *GPIO* para alimentar *LEDs* e um botão. Como se pode ver pelos comentários (linhas 177 a 181), os *LEDs* e o botão têm funções específicas. O primeiro *LED* liga assim que o processo de leitura inicia. O segundo, quando o programa começa a copiar os ficheiros, e os terceiro e quarto dependem de como corre o processo. Caso corra tudo como esperado, acende o LED 3, caso contrário entra no *loop While* (linhas 207 a 221), que verifica se o botão é pressionado. Enquanto o botão não for pressionado, o LED vermelho acende intermitentemente como forma de aviso, mas assim que o botão é pressionado, termina a função, e retorna à função *main()*.

```
231 def main():
232     mod_check()
233     setup()
234     while 1 > 0:
235         import RPi.GPIO as GPIO
236         import time
237         GPIO.setmode(GPIO.BCM)
238         GPIO.setup(7, GPIO.OUT) # LED 1: Ligado
239         GPIO.setup(8, GPIO.OUT) # LED 2: A trabalhar
240         GPIO.setup(10, GPIO.OUT) # LED 3: Sucesso
241         GPIO.setup(11, GPIO.OUT) # LED 4: Erro
242         GPIO.output(7, False)
243         GPIO.output(8, False)
244         GPIO.output(10, False)
245         GPIO.output(11, False)
246 
247         exif_read()
248 main()
```

De volta à função *main()*, podemos ver que a função *exif_read()*, onde estávamos antes, está envolvida num loop *while* infinito (pois verifica uma condição que será sempre verdadeira, 1 ser maior que 0). Neste *loop*, o script reinicia os *LEDs* a 0, e inicia a função novamente. Isto faz com que o processo principal esteja sempre a correr em segundo plano.

O circuito GPIO é o seguinte:



Este circuito dá uso a uma resistência de 820Ω, que controla a corrente que chega dos 10v (provenientes das portas *GPIO* 2 e 4, cada uma distribuindo 5v) aos 4 *LEDs*, ligados em paralelo, que por sua vez se ligam às portas GPIO respetivas. Assim, estabelece-se controlo entre o *Raspberry PI* e os *LEDs*.

O botão encontra-se desconectado do resto do circuito, sendo apenas uma ligação direta entre uma porta *GPIO* que está a ser lida como entrada e outra que serve de *ground.*

Conclusão

Com a concretização desta PAP, consegui aprofundar tanto os meus conhecimentos de programação, através da linguagem *Python* e dos protótipos iniciais em C*++,* bem como os meus conhecimentos de eletrónica, e ainda sobre as tecnicidades da fotografia digital.

Enfrentei dificuldades, contudo com o estudo sobre a matéria, fosse através de pesquisas online ou até de livros, consegui concretizar todos os objetivos a que me havia proposto.

Através da realização deste relatório, também obtive prática no domínio de discurso técnico, mas explícito, de modo que pessoas com menos conhecimentos consigam compreender o necessário, e pessoas dentro da área consigam apreender ainda mais.

Webgrafia

Estudos-base de programação python feitos em Codecademy:

https://www.codecademy.com

Simulador de circuitos:

https://www.falstad.com/circuit/~

Imagens 3d criadas com Cinema 4D

https://www.maxon.net/en/cinema-4d/

Materiais obtidos em:

https://mauser.pt/catalog/

https://www.pcdiga.com/

Documentação geral sobre o Raspberry Pi:

https://www.raspberrypi.org/documentation/

- Ficheiro rc.local
  - https://www.raspberrypi.org/documentation/

Especificações do Raspberry Pi 4

https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

Livrarias Python3 externas utilizadas:

colorama

https://pypi.org/project/colorama/

configparser 5.0.2

https://pypi.org/project/configparser/

pathlib

https://pypi.org/project/pathlib/

ExifRead 2.3.2

https://pypi.org/project/ExifRead/

Bibliografia

Sherz,Paul ; Simon Monk

(2016) Practical Electronics for Inventors, New York: MacGrawhill