Academia de Ensino Superior de Mafra
Cursos Técnicos Superiores Profissionais (CTESP)

ipt Instituto Politécnico de Tomar

GESTÃO DE UMA BASE DE DADOS SQL ATRAVÉS DE UM GUI EM PYTHON

AUTOR(ES):

FILIPE MIGUEL PEREZ MAGALHÃES SILVA FERREIRA - 81818

LUIZ FELLIPE DE PAULA SILVA SANTOS - 81810

PEDRO MIGUEL TELES MANUEL - 81817

RICARDO MIGUEL CURADO MILAGRE - 81750

DISCIPLINA/PROFESSOR:

PROGRAMAÇÃO AVANÇADA/ FERNANDO BARROS

ACADEMIA DE ENSINO SUPERIOR DE MAFRA, 23 de junho de 2022

ÍNDICE

1 - RESUMO/ABSTRACT

Foi proposto como projeto final a criação e implementação de um sistema de gestão de stocks/inventários de uma empresa/loja de componentes de computadores, com recurso à linguagem de programação Python com ligação a um sistema de gestão de base de dados, SQL Server. Este trabalho final tem uma componente de avaliação numa outra cadeira, Bases de Dados, daí a escolha e implementação do sistema de gestão de base de dados indicado.

Todo o algoritmo será explicado detalhadamente mais à frente neste relatório, sob forma de ser compreendido as ideias implementadas no projeto.

2 - INTRODUÇÃO

Neste relatório foi feita uma abordagem aos processos implementados em Python, que resultaram no sucesso da nossa GUI em realizar operações básicas numa base de dados. O nosso objetivo é dar a conhecer o nosso pensamento e arquitetura do programa, de forma que o mesmo seja percetível por outras pessoas. Para tal, as próximas páginas usam uma linguagem muito básica e de fácil entendimento mencionando, no entanto e sempre que necessários os conceito técnicos para melhor compreensão.

Para realização deste trabalho foi realizada pesquisa em diversos sites, sendo que o mais utilizado foi o stackoverflow. Foram também usados os slides dados em aula, bem como exercícios realizados em aula e outros exemplos.

Esperamos que após leitura deste relatório, seja claro para colegas de turma, professores bem como outras pessoas, ter uma noção básica dos diferentes métodos utilizados que nos permitiram concluir este trabalho prático. É feita uma desconstrução de cada tópico, de forma a poder ser mais fácil a compreensão do mesmo. Quanto as conclusões a que chegamos, carecem de ser confirmadas, mas, no geral, achamos que a interligação em GUI's e bases de dados é não só possível, mas recomendada para um melhor entendimento do funcionamento de um sistema de gestão de informação.

3 - DESCRIÇÃO DO PROBLEMA

O presente trabalho tem como intuito a manipulação de dados guardados numa base de dados criada em SQL Server, através da linguagem de programação Python. A par disto, estabeleceu-se como objetivo a extração de informação para ficheiros PDF e para páginas HTML.

Todas estas funcionalidades estão aplicadas num ambiente gráfico implementada também em Python.

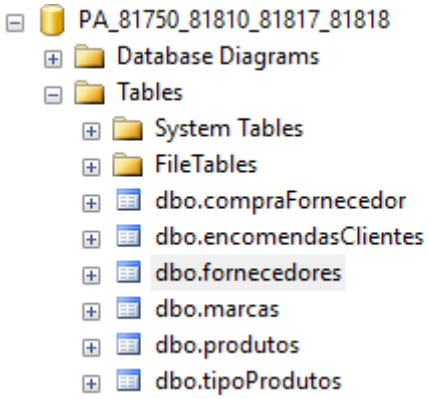
Para o acesso à base de dados, foi implementado um sistema de reconhecimento facial onde apenas os utilizadores com fotografias no diretório, e consequentemente, com permissões ao servidor, poderão afetar e/ou visualizar a informação presente na base de dados.

Deste modo, o presente relatório demonstrará os caminhos e alternativas adotadas para a melhor aplicação das funcionalidades descritas acima para o propósito estabelecido aquando da apresentação do guião.

4 - DESENVOLVIMENTO

4.1 - Definição da base de dados, tabelas e campos/Ligação entre Pycharm e SQL Server

Numa primeira instância, foi criada uma base de dados num servidor (criado na cadeira de Bases de Dados), onde foram definidas as tabelas a serem usadas e respetivos campos com os devidos tipos de dados.



Repare-se então que as tabelas criadas foram “compraFornecedor”,”encomendasClientes”, ”fornecedores”, “marcas”, “produtos” e “tipoProdutos”. De realçar que devido a problemas temporais, as tabelas “compraFornecedor” e “encomendasClientes” não foram usadas no projeto, ainda que possam vir a ser implementadas sob forma de melhorar o propósito do projeto.

Tabela **produtos**

idProduto	int
nomeProduto	nvarchar(255)
stockComprado	int
stockVendido	int
stockAtual	
stockSeguranca	
idMarca	int
idTipoProduto	int
idFornecedor	int

Na coluna à esquerda, estão presentes os nomes dos campos e à direita o respetivo tipo de dados. “nomeProduto” será do tipo string enquanto o stockAtual e o stockSeguranca estão

predefinidas no próprio SQL Server, isto é, stockAtual = stockComprado – stockVendido, e stockSeguranca = 5. Este campo, stockSeguranca, é importante no sentido de impor a condição de extração de informação para PDF, isto é, caso o stockAtual seja inferior ao stockSegurança, será emitido um alerta em PDF de iminência de rotura de stock (será explicado detalhadamente mais à frente.

Tabela fornecedores

idFornecedor	int
nomeFornecedor	nvarchar(255)
contacto	bigint
email	nvarchar(255)

Tabela marcas

idMarca	int
nomeMarca	nvarchar(255)

Tabela tipoProdutos

idTipoProduto	int
tipoProduto	nvarchar(255)

As tabelas “fornecedores”,”marcas” e “tipoProdutos” têm todas chaves primárias que irão, por ventura, estar na tabela “produtos” como chaves estrangeiras de forma a que haja uma relação entre estas.

Para estabelecer a conexão entre o software de desenvolvimento de Python (Pycharm) e o SQL Server foi utilizada a biblioteca *pymssql*.

```
import pymssql
```

```
#CONEXÃO AO SQL SERVER
mydb = pymssql.connect('srvsql-ipt.ddns.net', '81818', '81818', "PA_81750_81810_81817_81818")
cursor = mydb.cursor(as_dict=True)
```

Faz-se assim o pymssql.connect sendo que os argumentos passados são, respetivamente, o nome/IP do servidor, o user, a password e a base de dados a que queremos ter acesso. A seguir à conexão, iniciamos um cursor de forma que seja possível aceder a toda a informação dentro da base de dados escolhida.

4.2 – Reconhecimento Facial

Como referido anteriormente, foi estabelecido como objetivo a implementação de um sistema de reconhecimento facial com o intuito de fazer login à base de dados criada. Para o efeito, foi criado um módulo chamado ***face_.*** onde são importadas as seguintes bibliotecas:

- cv2;
- numpy;
- face_recognition;
- os;
- datetime;
- pymmsql.

```
import cv2
import numpy as np
import face_recognition
import os
from datetime import datetime
import pymssql
```

```
def facerecog():
    path='imagem'
    fotos=[]
    classNomes=[]
    myList=os.listdir(path)

    for cl in myList:
        imgAtual=cv2.imread(f'{path}/{cl}')
        fotos.append(imgAtual)
        classNomes.append(os.path.splitext(cl)[0])
    print("\nNOMES PRESENTES NA BASE DE DADOS")
    print (classNomes)

    def descobrir(fotos):
        encodeLista=[]
        for img in fotos:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            encode = face_recognition.face_encodings(img)[0]
            encodeLista.append(encode)
        return encodeLista
```

A função ***facerecog()*** indica o diretório onde se encontram as fotografias dos membros que estão autorizados a fazer a conexão à base de dados. Dentro desta função, existe uma outra função chamada ***descobrir(fotos)*** que passa como parâmetro a variável "fotos", isto é, a lista (ou pasta) e onde irá percorrer essa mesma pasta para selecionar a fotografia "desejada".

```
def arquivar(nome):
    with open('Arquivo.csv','r+') as f:
        myDataList=f.readline()
        nameList= []
        for line in myDataList:
            entry = line.split(',')
            nameList.append(entry[0])
        if nome not in nameList:
            now = datetime.now()
            dtString = now.strftime('%H:%M:%S')
            f.writelines(f'\n{nome},{dtString}')
```

A função ***arquivar(nome)*** também inserida dentro da função ***facerecog()*** servirá para criar um ficheiro do tipo .csv, em modo leitura e escrita, que servirá como uma espécie de log info, isto é, serão registados todos os logins efetuados na base de dados em tempo real.

```
RICARDO,19:04:57
RICARDO,19:05:40
LUIZ,19:07:31
LUIZ,19:09:44
LUIZ,19:10:34
RICARDO,19:12:00
RICARDO,19:12:47
RICARDO,19:15:04
RICARDO,19:16:11
RICARDO,19:17:51
RICARDO,19:20:44
RICARDO,19:24:28
RICARDO,19:25:29
RICARDO,19:27:30
RICARDO,19:34:57
RICARDO,19:36:23
RICARDO,19:45:17
PEDRO,19:46:03
RICARDO,19:55:34
RICARDO,15:16:59
RICARDO,15:28:34
RICARDO,15:29:38
```

```
encodeListaConhecidos=descobrir(fotos)
print ("\nCODIFICAÇÃO COMPLETA")

cap = cv2.VideoCapture(0)

print("PRONTO A RECEBER CARAS")
login=""
while True:
    sucess, img=cap.read()
    imgS=cv2.resize(img,(0,0),None,0.25,0.25)
    imgS= cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)

    faceWebcam = face_recognition_models.face_locations(imgS)
    encodeWebcam = face_recognition_models.face_encodings(imgS,faceWebcam)
    controlo = True
    for encodeFace,faceLoc in zip(encodeWebcam,faceWebcam):
        iguais = face_recognition_models.compare_faces(encodeListaConhecidos,encodeFace)
        faceDis = face_recognition_models.face_distance(encodeListaConhecidos,encodeFace)
        iguaisIndex = np.argmin(faceDis)
```

```
        if iguais[iguaisIndex]:
            nome = classNomes[iguaisIndex].upper()
            login = classNomes[iguaisIndex].upper()
            y1,x2,y2,x1=faceLoc
            y1, x2, y2, x1 = y1*4,x2*4,y2*4,x1*4
            cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)
            cv2.rectangle(img,(x1,y2-35),(x2,y2),(0,255,0),cv2.FILLED)
            cv2.putText(img,nome,(x1+6,y2-6),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
            arquivar(nome)
            controlo = False
    cv2.imshow('Webcam',img)
    cv2.waitKey(1)
    if controlo==True:
        continue
    elif controlo==False:
        cv2.destroyAllWindows()
        break
```

As duas imagens anteriores indicam o posicionamento da janela que se abrirá para fazer o reconhecimento facial, bem como a posição dos “quadrados” que irão aparecer sob as faces após reconhecer esta mesma, indicando o nome da pessoa que estará a fazer login.

```
if (login !=""):
    if (login =="RICARDO"):
        Entrar("Ricardo")
    elif (login =="PEDRO"):
        Entrar("Pedro")
    elif (login =="FILIPE"):
        Entrar ("Filipe")
    elif (login =="LUIZ"):
        Entrar ("Luiz")
    elif (login =="FERNANDO"):
        Entrar ("Fernando")
```

Caso o login seja um dos nomes acima, irá ser chamada a função ***Entrar(x).***

```
def Entrar(x):
    if (x=="Ricardo"):
        mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
        cursor = mydb.cursor(as_dict=True)
        nome=x
        for x in cursor:
            print(x)
        mydb.close()
    elif (x=="Pedro"):
        mydb = pymssql.connect('srvsql-ipt.ddns.net', '81817', '81817', "PA_81750_81810_81817_81818")
        cursor = mydb.cursor(as_dict=True)
        nome = x
    elif (x=="Filipe"):
        mydb = pymssql.connect('srvsql-ipt.ddns.net', '81818', '81818', "PA_81750_81810_81817_81818")
        cursor = mydb.cursor(as_dict=True)
        nome = x
    elif (x=="Luiz"):
        mydb = pymssql.connect('srvsql-ipt.ddns.net', '81810', '81810', "PA_81750_81810_81817_81818")
        cursor = mydb.cursor(as_dict=True)
        nome = x
```

Esta função fará então a conexão com a base de dados consoante a cara reconhecida, isto é, fará o login com o user e password associados à cara reconhecida.

4.3 – MenuDB()

4.3.1 – Bibliotecas utilizadas

O main code encontra-se neste ficheiro e será explicado detalhadamente e como se procedeu à sua divisão em funções. Antes disso, é importante referir as bibliotecas importadas:

- tkinter;
- pymssql

São ainda importados os módulos criados:

- *Reconhecimento* (reconhecimento facial)
- *PDF*
- *HTML*

```
from tkinter import *
import Reconhecimento
import pymssql
from PDF import LoadPDF
from HTML import printit
```

4.3.2 – Criação/Manipulação de PDFs

Foi criado um módulo denominado PDF, onde foram importadas as seguintes bibliotecas:

```
from fpdf import FPDF
import pymssql
import webbrowser
```

```python
def LoadPDF():
    class PDF(FPDF):
        pass
        def lines(self):
            self.set_line_width(0.0)
            self.line(5.0, 5.0, 205.0, 5.0)  # top one
            self.line(5.0, 292.0, 205.0, 292.0)  # bottom one
            self.line(5.0, 5.0, 5.0, 292.0)  # left one
            self.line(205.0, 5.0, 205.0, 292.0)  # right one
            self.rect(5.0, 5.0, 200.0, 287.0)
            self.rect(8.0, 8.0, 194.0, 282.0)
            self.set_xy(0.0, 0.0)
            self.set_font('Arial', 'B', 24)
            self.set_text_color(255, 20, 147)
            self.cell(w=210.0, h=40.0, align='C', txt="ROTURAS", border=0)
```

É inicializada uma função denominada de ***LoadPDF()***. Primeiramente, criou-se uma classe PDF onde é definida uma função ***lines()*** com um conjunto de métodos, que servirá para criar o template do PDF.

```python
pdf = PDF()
pdf.add_page()
pdf.lines()
pdf.set_font('Arial', '', 14)
pdf.set_text_color(0, 0, 255)
pdf.write(6, "\n\n\n\n\nEstes produtos estão em rotura: \n\n")
pdf.set_text_color(0, 0, 0)

cursor.execute('SELECT nomeProduto, stockAtual, stockSeguranca FROM produtos WHERE stockAtual < stockSeguranca')
listaRotura = []


for x in cursor:
    listaRotura.append(x)
```

É de seguida inicializado o objeto pdf onde serão chamados os métodos criados na função lines(). Após isto, cria-se uma querie que será o fator determinante para a criação e escrita no pdf com os artigos que irão estar em iminência de rotura.

```
for x in cursor:
    listaRotura.append(x)

pre = len(listaRotura)
i = 0
while i < pre:
    pdf.set_font('Arial', '', 14)
    pdf.set_text_color(0, 0, 0)
    pdf.write(10, listaRotura[i]['nomeProduto'] + '\nStock atual: ' + str(
        listaRotura[i]['stockAtual']) + '\n Stock Segurança: ' + str(listaRotura[i]['stockSeguranca']) + '\n\n')
    i += 1
mydb.close()
pdf.output('roturas.pdf', 'F')
webbrowser.open_new('roturas.pdf')
```

Irão ser adicionadas à lista "listaRotura" todos os registos que se enquadrem com o que foi definido na querie, isto é, todos os artigos com um stockAtual inferior ao stockSeguranca (5).

Essa lista será então escrita em pdf enquanto a variável i é inferior ao tamanho da listaRotura. Por fim, será feito o output desse pdf no ficheiro 'roturas.pdf', que será aberto no browser quando clicarmos no botão desenhado para o efeito.

4.3.3 – HTML

Foi idealizado a exportação de informação para uma página de HTML. Irão ser apresentados dados pertinentes, como o nome do artigo, os stocks atuais destes, os stocks vendidos e/ou comprados, bem como as marcas e fornecedores. Para o efeito, foram importadas as seguintes bibliotecas:

```
import webbrowser
import pymssql
import threading
import keyboard
```

De seguida foi definida a query apropriada a exportar os dados que queremos apresentar na página de html.

```
querys = (
    'SELECT nomeProduto,\'
    'stockComprado,\'
    'stockVendido,\'
    'tipoProduto,\'
    'nomeMarca,\ '
    'nomeFornecedor\'
    'FROM produtos\'
    'INNER JOIN tipoProdutos ON produtos.idTipoProduto = tipoProdutos.idTipoProduto\'
    'INNER JOIN marcas ON produtos.idMarca = marcas.idMarca\'
    'INNER JOIN fornecedores ON produtos.idFornecedor = fornecedores.idFornecedor'
    ')
```

Em relação à criação do html, temos um ciclo for que nos dá os dados da base de dados fazendo depois um append à lista vazia com o nome “listaSite”.

```
listaSite = []
cursor.execute(querys)
for x in cursor:
    listaSite.append(x)
```

Com a variável r1 descobrimos quantos objetos existem na “listaSite”

A seguir cria ou abre um ficheiro em html.

```
r1 = len(listaSite)
f = open('index.html','w')
```

Foi dividido o html em três partes, onde duas delas escrevem o html.

E uma para poder fazer um loop à variável r1 para ter todos os dados da variável, e enquanto a nossa variável de controlo “i” for menor que r1 insere os dados no html divididos.

```
for n in range(r1):
    while i < r1:
        messageValores += f"""
            <tr class="ocontent">
                <td class="produtos">{listaSite[i]['nomeProduto']}</td>
                <td class="ocontent">{listaSite[i]['tipoProduto']}</td>
                <td class="ocontent">{listaSite[i]['nomeFornecedor']}</td>
                <td class="ocontent">{listaSite[i]['nomeMarca']}</td>
                <td class="ocontent">{listaSite[i]['stockVendido']}</td>
                <td class="ocontent">{listaSite[i]['stockComprado']}</td>
                </tr>
            """

        i += 1
```

Por fim juntamos as 3 partes do html em uma só e fecha-se o ficheiro, escrevendo para o ficheiro “index.html”.

```
        messageFim = """
    </table>
    </body>
    </html>
    """
    message = messageHead + messageValores + messageFim
f.write (message)
f.close()
```

Temos uma função de abertura e o fecho do ficheiro de html na qual a package keyboard simula um input de keyboard(ctrl+w) de fecho de separador.~

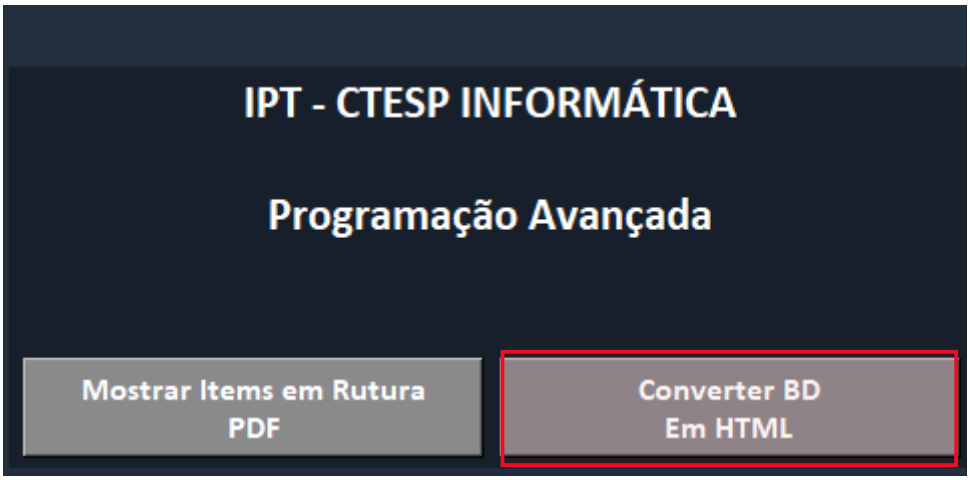
```
def botao_open(filename):
    keyboard.press_and_release('ctrl+w')  # closes the last tab
    webbrowser.open(filename)
```

Temos uma última função que com threading em 30 em 30 segundos fecha o ficheiro de html e abre um novo simulando assim um refresh da página.

Esta simulação de refresh foi feita desta maneira pois o python sem nenhuma library não é possivel fazer um refresh.

```
def printit():
  threading.Timer(30.0, printit).start()
  criarHtml()
  botao_open('index.html')
```

Ao clicar no botão “Converter BD para HTML”, será exportada para uma página HTML, como comprova os seguintes resultados:



Página HTML:

Produtos

Academia de Ensino Superior de Mafra

Produto	Tipo de Produto	Marca	Fornecedor	Stock Vendido	Stock Comprado
kjlhgv	Placas Gráficas	AMD	Outros	3	2
Ryzen 7 3800x PentaCore	Placas Gráficas	AMD	AMD	10	20
Ryzen 9 5900X 12-Core	Processadores	AMD	AMD	0	0
Core i9 -12500 16-core	Placas Gráficas	NVIDIA	Intel	0	10
Core i9-12900k 16-Core	Processadores	Intel	Intel	0	0
Core i7-12700k 12-Core	Processadores	Intel	Intel	0	0
Motherboard ATX MSI B550-A Pro	Motherboards	AMD	AMD	0	0
Motherboard Micro-ATX MSI B550M Pro-VDH WiFi	Motherboards	AMD	AMD	0	0
Motherboard Extended-ATX Asus ROG Zenith Extreme	Motherboards	AMD	AMD	7	0
Motherboard ATX MSI MAG Z690 Torpedo	Motherboards	Intel	Intel	0	0
Motherboard Micro-ATX MSI PRO B660M-A WiFI DDR4	Motherboards	Intel	Intel	0	0
Motherboard ATX MSI MEG Z690 Unify	Motherboards	Intel	Intel	0	0
XFX Radeon RX 6600 Speedster SWFT 210	Placas Gráficas	AMD	AMD	0	0
PowerColor Red Devil Radeon RX 6750XT	Placas Gráficas	AMD	AMD	0	0
MSI Radeon RX 6700 XT Mech 2X	Placas Gráficas	AMD	AMD	0	0
Gigabyte GeForce RTX 3060 Gaming	Placas Gráficas	NVIDIA	NVIDIA	0	0
Zotac Gaming GeForce RTX 3070 GDDR6X	Placas Gráficas	NVIDIA	NVIDIA	0	0
MSI GeForce RTX 3060 Gaming X 12G	Placas Gráficas	NVIDIA	NVIDIA	0	0
RAM G SKILL Trident Z5 32Gb DDR5	Memórias RAM	Banana	Outros	0	0
RAM Corsair Vengeance RGB RS 32GB DDR4	Memórias RAM	Banana	Outros	0	0
Corsair Dominator Platinum RGB 64GB DDR5	Memórias RAM	Banana	Outros	0	0
Seagate Barracuda 2TB SATA III	Discos Rigidos	Banana	Outros	0	0
SSD M.2 2280 Western Digital Black	Discos Rigidos	Banana	Outros	0	0
SSD m2. 2280 GIGABYTE M.30 TTB	Discos Rigidos	Banana	Outros	9	0
TESTE	Placas Gráficas	NVIDIA	AMD	2	2
HFJFJ	Placas Gráficas	NVIDIA	NVIDIA	2	1

4.3.4 – Funções Aninhadas

```python
def AdicionarBD(): # SABE QUAL A DB ABERTA NA FUNÇÃO ANTERIOR E FAZ APPEND À ABERTA
    if variavel.get() == "Produtos":
        AdicionarProdutos()
    elif variavel.get() == "Marcas":
        AdicionarMarcas()
    elif variavel.get() == "Fornecedores":
        AdicionarFornecedores()
    elif variavel.get() == "Tipo de Produtos":
        AdicionarTipoProdutos()
```

```python
def AdicionarBD_UPD(): # SABE QUAL A DB ABERTA NA FUNÇÃO ANTERIOR E FAZ O UPDATE À ABERTA
    if variavel.get() == "Produtos":
        UpdateProdutos()
    elif variavel.get() == "Marcas":
        UpdateMarcas()
    elif variavel.get() == "Fornecedores":
        UpdateFornecedores()
    elif variavel.get( ) == "Tipo de Produtos":
        UpdateTipoProdutos()
```

```python
def DeleteBD_UPD(): # SABE QUAL A DB ABERTA NA FUNÇÃO ANTERIOR E FAZ O DELETE À ABERTA
    if variavel.get() == "Produtos":
        DeleteProdutos()
    elif variavel.get() == "Marcas":
        DeleteMarcas()
    elif variavel.get() == "Fornecedores":
        DeleteFornecedores()
    elif variavel.get() == "Tipo de Produtos":
        DeleteTipoProdutos()
```

As imagens mostradas acima mostram um conjunto de funções aninhadas, isto é, funções dentro de uma função principal. Cada uma delas corresponde às três funcionalidades implementadas na parte gráfica de Adicionar, Atualizar e Eliminar registos na base de dados.

Cada função dentro dessas funções principais será explicada detalhadamente.

A variável “variavel” indica qual foi a tabela selecionada para a manipulação de dados, e consoante o que foi escolhido, chamará a função respetiva à tabela escolhida.

```
def confirmar(): # DEPENDENDO DO QUE O USER SELECIONAR EM BOTAO DE ESCOLHA, APRESENTA A DB.
    if variavel.get() == "Produtos":
        ListaProdutos()
        ChangeLog_Generic("Produtos")
    elif variavel.get() == "Marcas":
        ListaMarcas()
        ChangeLog_Generic("Marcas")
    elif variavel.get() == "Fornecedores":
        ListaFornecedores()
        ChangeLog_Generic("Fornecedores")
    elif variavel.get() == "Tipo de Produtos":
        ListaTipoProdutos()
        ChangeLog_Generic("Tipo de Produtos")
```

A função ***confirmar()*** corresponderá ao botão "Confirmar" da parte gráfica e que servirá para apresentar os resultados da tabela selecionada. Assim, dependendo da opção escolhida, ao clicar nesse botão Confirmar, irá ser chamada a função correspondente à tabela selecionada (ListaProdutos(), ListaFornecedores(), ListaMarcas(), ListaTipoProdutos().)

```
def ChangeLog_Generic(x): # CHANGELOG GENERICO QUE INDICA QUAL A DB ABERTA AO USER, BEM COMO OPÇÕES DISPONIVEIS
    tabela = x
    tabelaPrint = Label(ChangeLogTEXT, text=(
                "Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS - A apresentar resultados da tabela " + tabela +
                "\n\nSERVIÇOS DISPONIVEIS:\n1 - ADICIONAR: Insira um valor/item novo na BD (Não insira o id)\n2 - DELETE: Insira o valor do ID para este ser apagado"
                "\n3 - MODIFY: Insira o valor do ID do produto seguido dos campos presentes a alterar"),
                        anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
    tabelaPrint.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

```
def ChangeLog_Generic_NOVALUES(x):  # CASO O USER PRETENDA APPEND/DELETE/UPDATE SEM VALORES DA INDICAÇÃO DE ERRO
    tabela = x
    tabelaPrint = Label(ChangeLogTEXT, text=(
                "Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS - A apresentar resultados da tabela " + tabela +
                "\n\nDADOS NÃO INSERIDOS\n\nPor favor insira os dados e selecione uma opção"),
                        anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
    tabelaPrint.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

As funções ChangeLog serão chamadas e executadas sempre que houver alguma interação na parte gráfica. Desta forma, sempre que for selecionada uma tabela para a demonstração de dados, será impresso no ChangeLog que a conexão ao servidor foi realizada com sucesso e que está a apresentar os resultados relativamente à tabela selecionada. É também visivel no ChangeLog os serviços disponíveis, isto é, o que é que cada botão gráfico fará, como demonstra a imagem a seguir.

Logged in ao server srvsql-ipt.ddns.net
SERVIÇOS CONECTADOS - A apresentar resultados da tabela Produtos
SERVIÇOS DISPONIVEIS:
1 - ADICIONAR: Insira um valor/item novo na BD (Não insira o id)
2 - DELETE: Insira o valor do ID para este ser apagado
3 - MODIFY: Insira o valor do ID do produto seguido dos campos presentes a alterar

O ChangeLog mostrará mensagens sempre de acordo com a ação realizada. Por exemplo, se se eliminar um registo da tabela, a seguinte mensagem é apresentada:

Logged in ao server srvsql-ipt.ddns.net
SERVIÇOS CONECTADOS - A apresentar resultados da tabela Produtos
ENTRY NA TABELA Produtos FOI APAGADA

No caso de um update:

Logged in ao server srvsql-ipt.ddns.net
SERVIÇOS CONECTADOS - A apresentar resultados da tabela Produtos
ENTRY NA TABELA Produtos FOI UPDATED

4.3.5 – Definição das Queries e Funções para Apresentação de Resultados e Manipulação de Dados

FORNECEDORES

```
def ListaFornecedores():    # APRESENTA LISTA DE FORNECEDORES
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True) #CONNECT à DB
    querys = ['SELECT idFornecedor, nomeFornecedor,contacto,email FROM Fornecedores'] #QUERY
    cab = []        # RETIRA OS ELEMENTOS DO DICIONARIO UM A UM. CRIA 2 VARIAVEIS QUE GUARDAM OS NOMES NAS TABELAS
    tab = []        # E OS PRODUTOS
    cursor.execute(querys[0])
    for x in cursor:    #SEPARA OS KEYS DOS ARGUMENTOS.
        for key, value in x.items():
            cab.append(key)
            tab.append(str(value))
    cab = list(dict.fromkeys(cab))  # CODIGO PARA APAGAR REPETIDOS, DE FORMA A TERMOS APENAS UM LABEL DE KEYS.
    r1 = len(tab)                  #GUARDA TAMANHO DA TAB
    n = [n for n in range(0, r1, 4)] #  N É UMA LISTA QUE PRECORRE OS ELEMENTOS DE TAB, E QUE DE 4 EM 4 OS SEPARA
    f1 = list()                     # NESTE CASO FORAM 4, MAS O CÓDIGO É DINAMICO E É UTILIZADO NAS OUTRAS LISTAS.
    for item in n:                  # APENAS É NECESSÁRIO MUDAR O PASSO.
        v = tab[item:item + 4]      # V VAI TER  A UNIÃO DE ITEMS 4 A 4. (DINAMICO NOVAMENTE, USADO EM TODOS OS RESTANTES)
        f1.append(v)                #F1 VAI TER COMO RESULTADO APENAS OS VALUES ORIGINAIS.
    ViewerTEXTscroll = Scrollbar(ViewerTEXT)    # CRIAÇÃO DE SCROLLBAR QUE VARIA DE ACORDO COM TAMANHO DA LISTA
    ViewerTEXTscroll.pack(side=RIGHT,fill=Y)
    listbox = Listbox(ViewerTEXT, yscrollcommand=ViewerTEXTscroll.set)
    listbox.insert(END,str(cab))
    for a in f1:                    # INSERE CADA ELEMENTO NA LISTBOX.
        listbox.insert(END,a)
    listbox.place(x=5,y=5,width=760, height=775)    #COLOCAÇÃO DA LISTBOX
    listbox.config(width=500,justify=LEFT,)
    ViewerTEXTscroll.config(command=listbox.yview)
```

O código acima utiliza o comando (função **ListaFornecedores()**) SELECT de SQL para apresentar os dados presentes na tabela Fornecedores. O primeiro ciclo for percorre o cursor, que no

fundo é uma lista de dicionários, e o segundo ciclo for percorrerá cada dicionário dessa lista (cursor). Por cada iteração, irá fazer append às listas vazias criadas (cab e tab), onde na primeira serão adicionadas as tags do dicionário, ou seja, a key, e na segunda serão adicionadas os valores correspondentes a cada tag. Exemplo:

{nomeFornecedor : Intel} → key = nomeFornecedor; value = Intel

A lista cab[] ficará com a key e a lista tab[] ficará com o value.

De seguinda, indicamos que a lista cab não ficará com elementos repetidos *(cab = list(dict.fromkeys(cab))*, caso contrário, por cada registo que tivéssemos, iria ser impresso a key o mesmo número de vezes, isto é, caso tivéssemos 5 registos, iria aparecer "nomeFornecedor" cinco vezes.

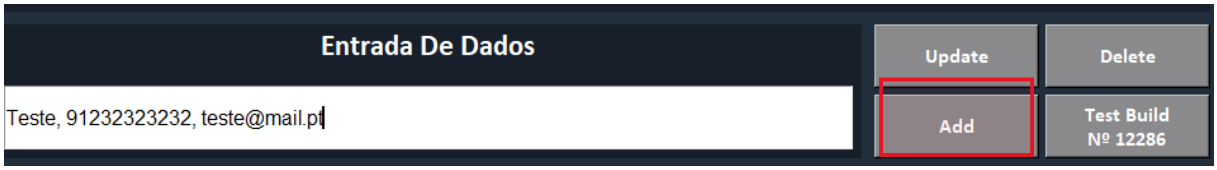
Posteriormente, declaramos e inicializamos uma variável r1 com o tamanho da lista tab. A saber o tamanho desta lista, usando compressão de listas, criamos uma lista n com o tamanho específico, para ser usada no ciclo for a seguir. Este último ciclo for irá fazer um append à lista f1 dos elementos da lista tab (sem elementos repetidos).

```
def AdicionarFornecedores():    # ADICIONA FORNECEDORES
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81817', '81817', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "INSERT INTO fornecedores(nomeFornecedor,contacto,email) VALUES (%s,%d,%s)" #QUERY
    if (ENTRADA2.get()==""):    # SE ENTRADA FOR VAZIA, REDIRECIONA PARA CHANGELOG ERRO. USADO EM TODAS AS FUNÇÕES APPEND,DELETE,MODIFY
        ChangeLog_Generic_NOVALUES("Fornecedores")
        breakpoint
    else:
        val = []
        for x in (ENTRADA2.get()).split(","): # CADA VEZ QUE DETECTA UMA ',' RECONHECE COMO SENDO UM NOVO PARAMETRO.
            val.append(x)                     # CADA VEZ QUE TEM ESSE PARAMETRO FAZ APPEND.
        val2 = (str(val[0]), int(val[1]), str(val[2]))
        try:
            if len(val) > 3:       # EXCEPÇÕES USADAS USADOS EM TODOS AS FUNÇÕES DE UPDATE/APPEND/DELETE. SERVEM PARA COMUNICAR AO USER NO CHANGLOG
                imprimir = Label(ChangeLogTEXT, # ERROS OBTIDOS. CASO SEJA REALIZADO COM SUCESSO TAMBÉM É COMUNICADO AO USER.
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            elif len(val) == 3:
                cursor.execute(sql, val2) #EXECUTA EM SQL
                mydb.commit() #NECESSARIO COMMIT PARA REALIZAR ALTERAÇÃO. SEM ISSO NÃO GRAVA MUDANÇAS.
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nADICIONADO À BASE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

A função **AdicionarFornecedores()** tem como funcionalidade o INSERT de novos fornecedores na base de dados, tal como se pode comprovar com a variável sql. A lista val terá posteriormente os valores introduzidos na parte gráfica. De seguida, fazemos corresponder cada índice da lista val à tupla val2, de forma a corresponder às variáveis definidas na querie. Faz-se de seguida um controlo de exceções com um *try-except.* Caso a lista val tenha um tamanho superior ao número de dados que suporta, irá ser mostrada uma mensagem de erro no ChangeLog, informando que houve uma falha a fazer a adição de dados, sendo que a querie não será executada. Caso a lista val tenha o tamanho exato de dados introduzidos (neste caso, 3), a querie será executada e será guardada a informação na base dados. Finalmente, o except tratará de erros como por exemplo a falha da conexão à base de dados e não irá ser possível manipular a mesma.

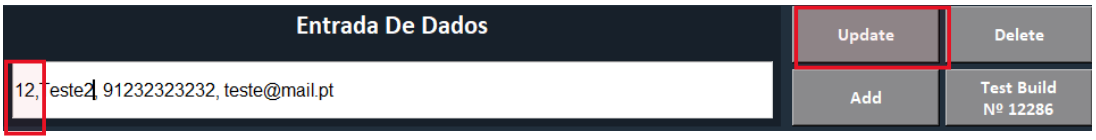
De realçar que os dados introduzidos para INSERT ou UPDATE são sempre separados por vírgulas, sendo que tal é tratado no código com o ENTRADA2.get().split(‘,’).

Exemplo:



```python
def UpdateFornecedores(): # UPDATE FORNECEDORES
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "UPDATE fornecedores SET nomeFornecedor=%s,contacto=%d,email=%s WHERE idFornecedor = %d"
    if (ENTRADA2.get()==""):
        ChangeLog_Generic_NOVALUES("Fornecedores")
        breakpoint
    else:
        val = []
        for x in (ENTRADA2.get()).split(","):
            val.append(x)
        val2 = (str(val[1]), int(val[2]), str(val[3]), int(val[0]))
        try:
            if len(val)>4:
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            elif len(val) == 4:
                cursor.execute(sql, val2)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nALTERADA NA BASE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

A função **UpdateFornecedores()** servirá para fazer a atualização de registos que já se encontrem na base de dados através da execução da querie UPDATE. A procura do registo a atualizar será feita através do id, neste caso, do fornecedor. Para conhecimento dos ids, é necessário selecionar a tabela e clicar em confirmar para ser apresentada a informação na janela Viewer. Todo o restante trecho de código da função segue exatamente a mesma lógica explicada na função **AdicionarFornecedores()**.

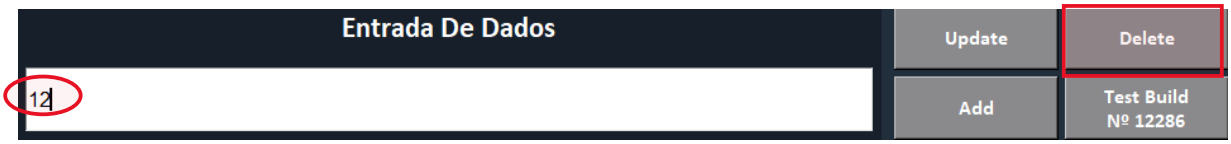


Neste caso, queremos fazer update do registo com o id = 12, em que o nomeFornecedor será alterado de 'Teste' para 'Teste2', como se pode comprovar na imagem seguinte.

```python
def DeleteFornecedores():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "DELETE FROM fornecedores WHERE idFornecedor = %d"
    if (ENTRADA2.get()==""):    # SE ENTRADA FOR VAZIA, REDIRECIONA PARA CHANGELOG ERRO
        ChangeLog_Generic_NOVALUES("Fornecedores")
        breakpoint
    else:
        val = ENTRADA2.get()
        try:
            if val == "":
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            else:
                cursor.execute(sql, val)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nALTERADA NA BASE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

No caso da função **DeleteFornecedores(),** irá ser apagado todo o registo com base no id selecionado. A estrutura try-except difere no sentido em que se não for adicionado nada no campo de entrada, irá mostrar erro no ChangeLog.

Exemplo: Vamos eliminar o registo adicionado anteriormente, ou seja, o fornecedor de id = 12.



Ao selecionar a tabela Fornecedores novamente, visualizamos o seguinte:

Como podemos observar, o id 12 foi eliminado.

PRODUTOS

A lógica implementada nas funções relativamente à tabela fornecedores é a mesma para as restantes tabelas, logo a explicação para as próximas funções será breve.

```
def ListaProdutos():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81817', '81817', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    querys = ['SELECT idProduto, nomeProduto, stockComprado, stockVendido,idMarca,idTipoProduto,idFornecedor FROM produtos']
    cab = []
    tab = []  #JÁ EXPLICADO
    cursor.execute(querys[0])
    for x in cursor:
        for key, value in x.items():
            cab.append(key)
            tab.append(str(value))
    cab = list(dict.fromkeys(cab))
    r1 = len(tab)
    n = [n for n in range(0, r1, 7)]
    f1 = list()
    for item in n:
        v = tab[item:item + 7]
        f1.append(v)
    ViewerTEXTscroll = Scrollbar(ViewerTEXT)
    ViewerTEXTscroll.pack(side=RIGHT,fill=Y)
    listbox = Listbox(ViewerTEXT, yscrollcommand=ViewerTEXTscroll.set)
    listbox.insert(END,str(cab))
    for a in f1:
        listbox.insert(END,a)
    listbox.place(x=5,y=5,width=760, height=775)
    listbox.config(width=500,justify=LEFT,)
    ViewerTEXTscroll.config(command=listbox.yview)
```

Função **ListaProdutos()** que irá fazer o select da tabela produtos e apresentar os resultados na Viewer box.

```
def AdicionarProdutos():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "INSERT INTO produtos(nomeProduto,stockComprado,stockVendido,idMarca,idTipoProduto,idFornecedor) VALUES(%s,%d,%d,%d,%d,%d)"
    val = []
    if (ENTRADA2.get()==""):    # SE ENTRADA FOR VAZIA, REDIRECIONA PARA CHANGELOG ERRO
        ChangeLog_Generic_NOVALUES("Produtos")
        breakpoint
    else:
        for x in (ENTRADA2.get()).split(","):
            val.append(x)
        val2 = (str(val[0]), int(val[1]), int(val[2]), int(val[3]), int(val[4]), int(val[5]))
        try:
            if len(val) > 6:
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            elif len(val) == 6:
                cursor.execute(sql, val2)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nADICIONADO À BASSE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\nSERVIÇOS CONECTADOS\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

A função **AdicionarProdutos()** irá fazer o INSERT de novos produtos à tabela produtos.

Lógica implementada igual à da função **AdicionarFornecedores()** com a querie devidamente alterada.

```
def UpdateProdutos():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "UPDATE produtos SET nomeProduto=%s,stockComprado=%d,stockVendido=%d,idMarca=%d,idTipoProduto=%d,idFornecedor=%d WHERE idProduto = %d"
    val = []
    if (ENTRADA2.get()==""):
        ChangeLog_Generic_NOVALUES("Produtos")
        breakpoint
    else:
        for x in (ENTRADA2.get()).split(","):
            val.append(x)
        val2 = (str(val[1]), int(val[2]), int(val[3]), int(val[4]), int(val[5]), int(val[6]), int(val[0]))
        try:
            if len(val)>7:
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            elif len(val) == 7:
                cursor.execute(sql, val2)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nALTERADA NA BASE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

A função **UpdateProdutos()** fará o UPDATE de produtos que já se encontrem na base de dados, seguindo exatamente a mesma lógica da função **UpdateFornecedores()**, onde se indica o id do produto que se quer modificar.

```
def DeleteProdutos():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "DELETE FROM produtos WHERE idProduto = %d"
    if (ENTRADA2.get()==""):
        ChangeLog_Generic_NOVALUES("Produtos")
        breakpoint
    else:
        val = ENTRADA2.get()
        try:
            if val == "":
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            else:
                cursor.execute(sql, val)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nALTERADA NA BASE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

A função **DeleteProdutos()** fará o DELETE de produtos já existentes na BD e para a eliminação de um registo, introduz-se primeiramente o id do produto a ser eliminado.

TIPOPRODUTOS

```
def ListaTipoProdutos():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81817', '81817', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    querys = ['SELECT idTipoProduto,tipoProduto FROM tipoProdutos']
    cab = []
    tab = []
    cursor.execute(querys[0])
    for x in cursor:
        for key, value in x.items():
            cab.append(key)
            tab.append(str(value))
    cab = list(dict.fromkeys(cab))
    r1 = len(tab)
    n = [n for n in range(0, r1, 2)]
    f1 = list()
    for item in n:
        v = tab[item:item + 2]
        f1.append(v)
    ViewerTEXTscroll = Scrollbar(ViewerTEXT)
    ViewerTEXTscroll.pack(side=RIGHT,fill=Y)
    listbox = Listbox(ViewerTEXT, yscrollcommand=ViewerTEXTscroll.set)
    listbox.insert(END,str(cab))
    for a in f1:
        listbox.insert(END,a)
    listbox.place(x=5,y=5,width=760, height=775)
    listbox.config(width=500,justify=LEFT,)
    ViewerTEXTscroll.config(command=listbox.yview)
```

Função **ListatTipoProdutos()** fará o SELECT dos dados guardados na tabela TipoProdutos.

```
def AdicionarTipoProdutos():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "INSERT INTO tipoProdutos(tipoProduto) VALUES (%s)"
    if (ENTRADA2.get()==""):
        ChangeLog_Generic_NOVALUES("Tipo de Produtos")
        breakpoint
    else:
        val2 = str(ENTRADA2.get())
        try:
            if val2 == "":
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            elif val2 != "":
                cursor.execute(sql, val2)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nADICIONADO À BASSE DE DADOS COM SUCESSO"
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

Função **AdicionarTipoProdutos()** fará o INSERT na tabela TipoProdutos. Mais uma vez, mesma lógica nas funções Adicionar.

```python
def UpdateTipoProdutos():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "UPDATE tipoProdutos SET tipoProduto=%s WHERE idTipoProduto = %d"
    val = []
    if (ENTRADA2.get()==""):
        ChangeLog_Generic_NOVALUES("Tipo de Produtos")
        breakpoint
    else:
        for x in (ENTRADA2.get()).split(","):
            val.append(x)
        val2 = (str(val[1]), int(val[0]))
        try:
            if len(val)>2:
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            elif len(val) == 2:
                cursor.execute(sql, val2)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nALTERADA NA BASE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

Função **UpdateTipoProdutos()** fará o UPDATE de registos já existentes na tabela correspondente. Novamente, seleciona-se o id que se quer alterar e de seguida, introduz-se os campos a alterar.

```python
def DeleteTipoProdutos():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "DELETE FROM tipoProdutos WHERE idTipoProduto = %d"
    if (ENTRADA2.get()==""):
        ChangeLog_Generic_NOVALUES("Tipo de Produtos")
        breakpoint
    else:
        val = ENTRADA2.get()
        try:
            if val == "":
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            else:
                cursor.execute(sql, val)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nALTERADA NA BASE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

Função **DeleteTipoProdutos()** fará o DELETE de registos presentes na tabela.

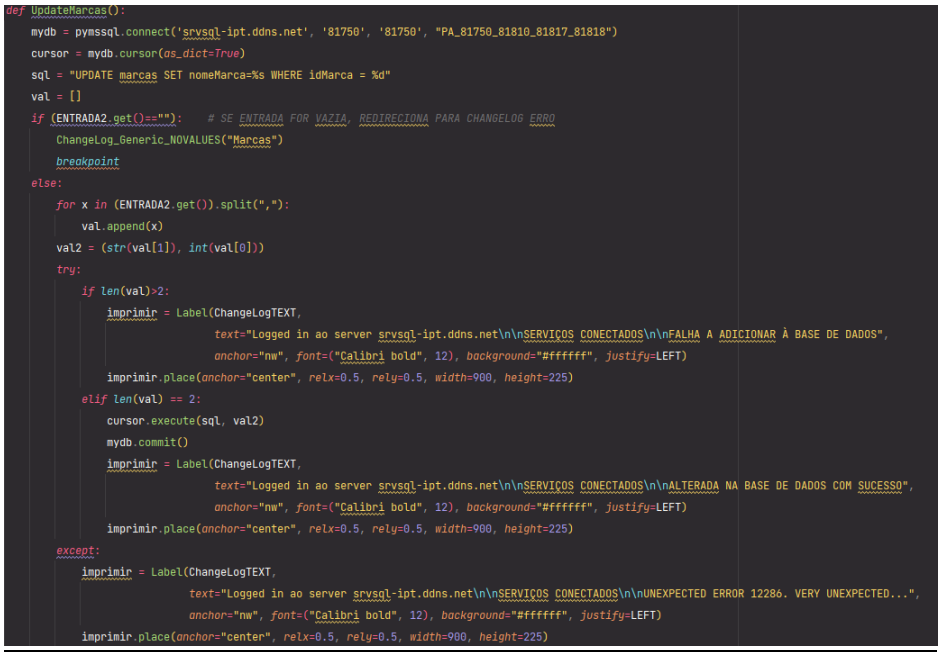
MARCAS

```
def ListaMarcas():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81817', '81817', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    querys = ['SELECT idMarca,nomeMarca FROM Marcas']
    cab = []
    tab = []
    cursor.execute(querys[0])
    for x in cursor:
        for key, value in x.items():
            cab.append(key)
            tab.append(str(value))
    cab = list(dict.fromkeys(cab))
    r1 = len(tab)
    n = [n for n in range(0, r1, 2)]
    f1 = list()
    for item in n:
        v = tab[item:item + 2]
        f1.append(v)
    ViewerTEXTscroll = Scrollbar(ViewerTEXT)
    ViewerTEXTscroll.pack(side=RIGHT,fill=Y)
    listbox = Listbox(ViewerTEXT, yscrollcommand=ViewerTEXTscroll.set)
    listbox.insert(END,str(cab))
    for a in f1:
        listbox.insert(END,a)
    listbox.place(x=5,y=5,width=760, height=775)
    listbox.config(width=500,justify=LEFT,)
    ViewerTEXTscroll.config(command=listbox.yview)
```

Função **ListaMarcas()** que fará o SELECT de todos os registos presentes na tabela marcas.

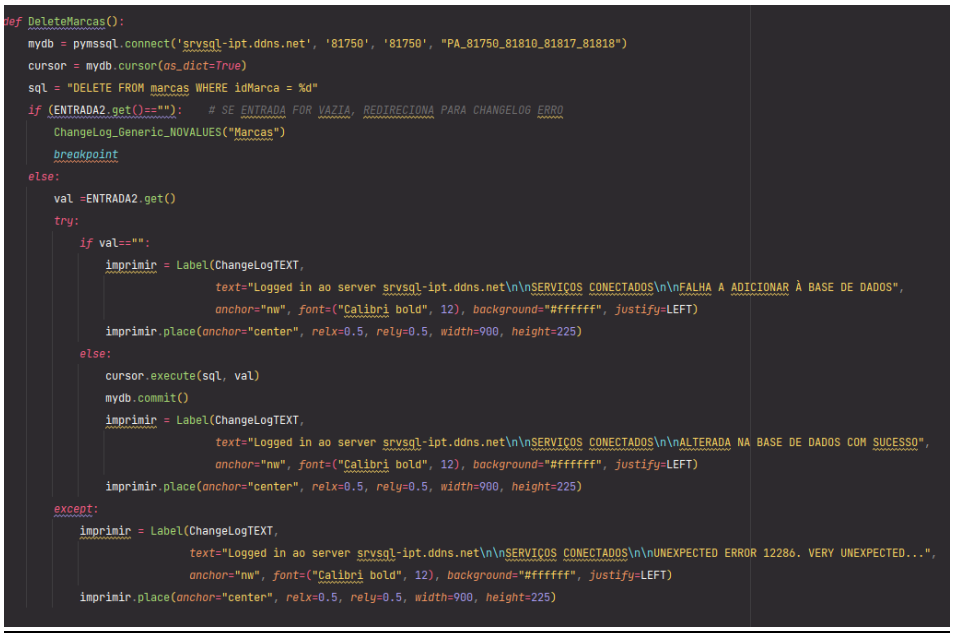
```
def AdicionarMarcas():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "INSERT INTO marcas(nomeMarca) VALUES (%s)"
    if (ENTRADA2.get()==""):    # SE ENTRADA FOR VAZIA, REDIRECIONA PARA CHANGELOG ERRO
        ChangeLog_Generic_NOVALUES("Marcas")
        breakpoint
    else:
        val2 = str(ENTRADA2.get())
        try:
            if val2 == "":
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            elif val2 != "":
                cursor.execute(sql, val2)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nADICIONADO À BASSE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

Função **AdicionarMarcas()** fará o INSERT de novos registos na tabela marcas.



```
def UpdateMarcas():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "UPDATE marcas SET nomeMarca=%s WHERE idMarca = %d"
    val = []
    if (ENTRADA2.get()==""):    # SE ENTRADA FOR VAZIA, REDIRECIONA PARA CHANGELOG ERRO
        ChangeLog_Generic_NOVALUES("Marcas")
        breakpoint
    else:
        for x in (ENTRADA2.get()).split(","):
            val.append(x)
        val2 = (str(val[1]), int(val[0]))
        try:
            if len(val)>2:
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            elif len(val) == 2:
                cursor.execute(sql, val2)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nALTERADA NA BASE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

Função **UpdateMarcas()** fará o UPDATE a registos já existentes na tabela consoante o id indicado.



```
def DeleteMarcas():
    mydb = pymssql.connect('srvsql-ipt.ddns.net', '81750', '81750', "PA_81750_81810_81817_81818")
    cursor = mydb.cursor(as_dict=True)
    sql = "DELETE FROM marcas WHERE idMarca = %d"
    if (ENTRADA2.get()==""):    # SE ENTRADA FOR VAZIA, REDIRECIONA PARA CHANGELOG ERRO
        ChangeLog_Generic_NOVALUES("Marcas")
        breakpoint
    else:
        val =ENTRADA2.get()
        try:
            if val=="":
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nFALHA A ADICIONAR À BASE DE DADOS",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
            else:
                cursor.execute(sql, val)
                mydb.commit()
                imprimir = Label(ChangeLogTEXT,
                                 text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nALTERADA NA BASE DE DADOS COM SUCESSO",
                                 anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
                imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
        except:
            imprimir = Label(ChangeLogTEXT,
                             text="Logged in ao server srvsql-ipt.ddns.net\n\nSERVIÇOS CONECTADOS\n\nUNEXPECTED ERROR 12286. VERY UNEXPECTED...",
                             anchor="nw", font=("Calibri bold", 12), background="#ffffff", justify=LEFT)
            imprimir.place(anchor="center", relx=0.5, rely=0.5, width=900, height=225)
```

Função **DeleteMarcas()** fará o DELETE a registos já existentes na tabela consoante o id indicado.

4.3.6 – Interface Gráfica (GUI)

Com a biblioteca tkinter podemos desenvolver a interface gráfica do nosso programa.

Nas definições gerais da janela que vamos fazer para o nosso menu de log in, definimos uma janela de dimensão 1920x1080, o que nos dá uma janela HD. Com diversos comandos como o state, podemos definir a abertura em full screen. Como o nosso objetivo era identificar a nossa instituição, importamos da biblioteca PIL, ImageTk e Image. Definimos um frame no qual colocamos a imagem que vemos no menu de log in.

```
####                              DEFINIÇÃO DE JANELA DA FUNÇÃO TESTES2()                              ####
#########################################################################################################


def about(): # MENU GUI - DEFINIÇÃO ABOUT#################################################################
    nova_janela = Toplevel(janela) # ABRE A JANELA ABOUT EM TOP LEVEL
    nova_janela.title('About')
    nova_janela.geometry('300x300') # TAMANHO DA JANELA
    x=Label(nova_janela,text="Trabalho realizado por\n\n Ricardo Milagre\n Pedro Manuel\n Luiz Santos\n Filipe Ferreira")
    x.place(width=275,height=275,rely=0.5,relx=0.5)

janela = Tk()
janela.title("Logged in srvsql-ipt.ddns.net")
janela.iconbitmap("favicon.ico")
janela.geometry('1920x1080')
janela.state('zoomed')
frame = Frame(janela, width=1920, height=1080)

menubar = Menu(janela, background='#ffffff')
# FILE#####################################################################################################
file = Menu(menubar, tearoff=0, foreground='black')
file.add_separator()
file.add_command(label="Exit", command=janela.quit)
menubar.add_cascade(label="File", menu=file)
janela.config(menu=menubar)
# ABOUT####################################################################################################
sobre = Menu(menubar, tearoff=0, foreground='black')
sobre.add_separator()
sobre.add_command(label="Exit", command=janela.quit)
janela.config(menu=menubar)
```

Também definimos apenas um botão file com a opção de Log in na nossa database. Através dos comandos backgroud, foreground, anchor, relx, rely podemos controlar o posicionamento das labels, botões bem como os comandos que cada um tem quando são pressionados.

Também temos um comando no final do nosso file chamado exit que termina a aplicação.

```python
def LogIn():
    janela = Tk()
    janela.title('Log In Menu')
    janela.geometry('1920x1080')
    janela.state('zoomed')
    frame = Frame(janela, width=600, height=400)
    frame.pack()
    frame.place(anchor='center', relx=0.5, rely=0.5)
    img = ImageTk.PhotoImage(Image.open("recurso2.jpg"))
    label = Label(frame, image=img)
    label.pack()
    # FILE#########################################################################################################
    menubar = Menu(janela, background='#ff8800', foreground='black', activebackground='white', activeforeground='black')
    file = Menu(menubar, tearoff=0, foreground='black')
    file.add_command(label="Log In", command=lambda: [ReconhecimentoFacial(), janela.destroy(), MenuDataBase()])
    file.add_separator()
    file.add_command(label="Exit", command=janela.quit)
    menubar.add_cascade(label="File", menu=file)
    janela.config(menu=menubar)
    janela.mainloop()

LogIn()
```

Ao fazer log in o user inicia o software de reconhecimento facial, que por sua vez faz destroy da janela atual e inicia uma nova. Esta é a janela de conexão com a Database.

```python
#BOTAO DELETE
APAGAR_BOTAO = Button(Opções, text="Delete", foreground="#FFFFFF",
             font=("Calibri bold", 12), anchor='center',
             background="#8A8A8C",command=DeleteBD_UPD)
APAGAR_BOTAO.place(y=400, x=835, width=130, height=50)
#BOTÃO TEST BUILD
TEST_BUTTON = Button(Opções, text="Test Build\nN° 12286", foreground="#FFFFFF",
             font=("Calibri bold", 12), anchor='center',
             background="#8A8A8C",)
TEST_BUTTON.place(width=130, height=50, x=835, y=455)
#BOTAO UPDATE
ATUALIZAR_BOTAO = Button(Opções, text="Update", foreground="#FFFFFF",
             font=("Calibri bold", 12), anchor='center',
             background="#8A8A8C",command=AdicionarBD_UPD)
ATUALIZAR_BOTAO.place(width=130, height=50, x=700, y=400)
```

```
#NOTÃO ADD
ADICIONAR_BOTAO = Button(Opções, text="Add", foreground="#FFFFFF",
            font=("Calibri bold", 12), anchor='center',
            background="#8A8A8C",command=AdicionarBD)
ADICIONAR_BOTAO.place(width=130, height=50, x=700, y=455)
#BOTAO PDF
PDF = Button(PA,text="Mostrar Items em Rutura\nPDF", foreground="#FFFFFF",
               font=("Calibri bold", 12), anchor='center',
               background="#8A8A8C",command=LoadPDF)
PDF.place(width=230, height=50, x=5, y=143)
#BOTÃO HTML
HTML = Button(PA, foreground="#FFFFFF", text="Converter BD\nEm HTML",
               font=("Calibri bold", 12), anchor='center',
               background="#8A8A8C",command=printit)
HTML.place(width=230, height=50, x=243, y=143)
janela.mainloop()
```

Nesta janela, para alem dos labels que já tinham sido criados, foram também usados botões que permitem aplicar comandos de adicionar, apagar, modificar a base de dados.

Em todos os botões e labels foram aplicadas as mesmas definições a nível de fonts e de cores.

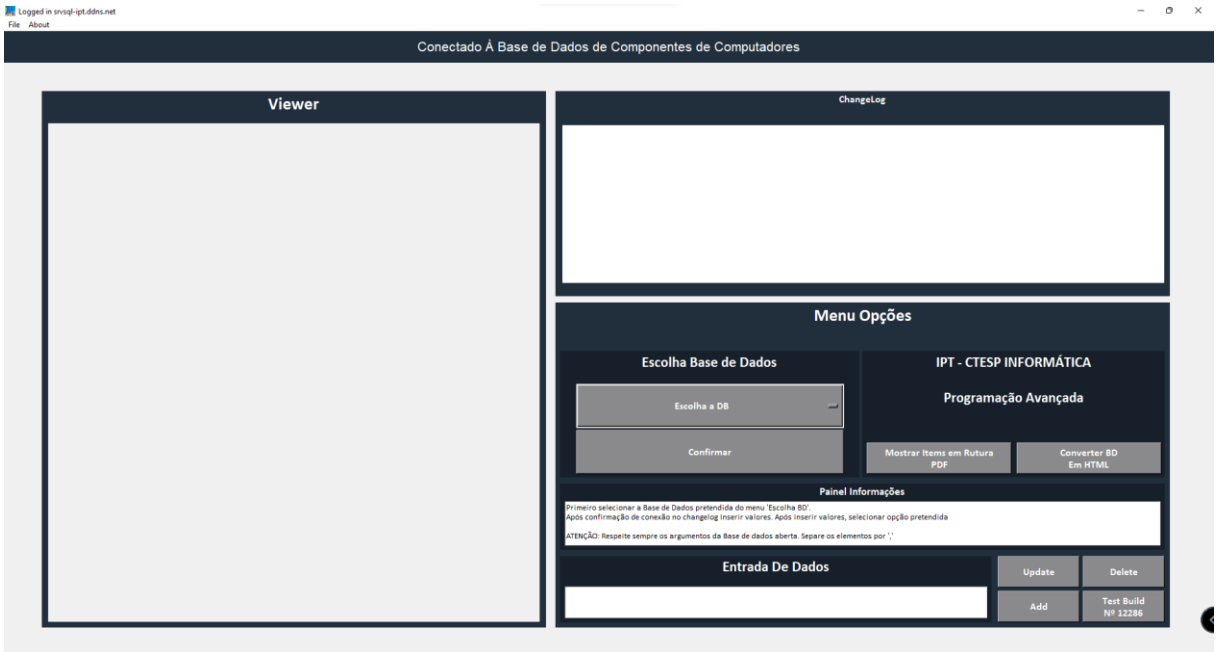
Em relação ao Changelog, Viewer e ao menu Opções, embora tenham sidos trabalhosos, são estativos e não requerem mudanças depois de estabelecidos.

```
ViewerTEXTscroll = Scrollbar(ViewerTEXT)    # CRIAÇÃO DE SCROLLBAR QUE VARIA DE ACORDO COM TAMANHO DA LISTA
ViewerTEXTscroll.pack(side=RIGHT,fill=Y)
listbox = Listbox(ViewerTEXT, yscrollcommand=ViewerTEXTscroll.set)
listbox.insert(END,str(cab))
for a in f1:                    # INSERE CADA ELEMENTO NA LISTBOX.
    listbox.insert(END,a)
listbox.place(x=5,y=5,width=760, height=775)    #COLOCAÇÃO DA LISTBOX
listbox.config(width=500,justify=LEFT,)
ViewerTEXTscroll.config(command=listbox.yview)
```

O mais complicado de realizar foi a criação de uma listbox dinâmica com scroll ativo. Para isso, foi definido que a dimensão teria de ser do 0 até ao comprimento da base de dados aberta pelo user. Para isso, criamos um ciclo for que vai desde 0 ate ao comprimento máximo da lista, que faz o insert de cada um linha a linha. Por fim fazemos o pack dentro do Viewer. A ativação de yscrollcomand permite a deslocação no eixo y, adaptando a lista ao tamanho da nossa Viewer box.

No final, obtemos uma GUI capaz de apresentar ao user uma janela de login e outra quando este se conecta à database.

O intuito de usar 2 GUI'S é eviar que pessoas sem permissões possam aceder as nossas tabela, manipulando e alterando valores sem permissões para tal.



5 - Conclusões

Estamos em condições de admitir que o projeto cumpre com os requisitos pedidos aquando da realização do mesmo, isto é, foram utilizados os principais pontos de matéria lecionados ao longo do semestre.

Porém, estamos cientes de que há alguns aspetos que necessitariam de retificação e melhoria, como por exemplo, a parte de atualizar registos, uma vez que não está tão *user-friendly* como gostaríamos que estivesse.

Apesar de ter sido estruturada uma divisão de tarefas entre os membros do grupo, todos estão por dentro de cada trecho de código e em condições de entender e explicar o que cada função/trecho realiza. Todo o programa e a sua lógica foram construídos passo-a-passo e de forma dinâmica.

O programa funciona, porém, não está 100% funcional, de acordo com os problemas já identificados.

Com isto, gostaríamos de agradecer a disponibilidade do professor pela ajuda na resolução de problemas que fomos encontrando ao longo da implementação do projeto.