LINGUAGENS DE PROGRAMAÇÃO

11 JANEIRO 2023

MINI PROJETO

PyChat

TIAGO MOTA
JOSÉ FERREIRA
RODRIGO SOUSA
MIGUEL CARVALHO

Índice

Índice de Imagens

1. Introdução

No presente documento abordamos o nosso mini projeto que consiste na criação de um chat em Python com o objetivo de proporcionar uma ferramenta de comunicação simples, utilizamos o conhecimento retido nas aulas de Linguagens de Programação e colmatamos com investigação. Ainda ao longo deste documento apresentamos um diagrama de classes, os testes e os resultados que obtivemos.

Para além da biblioteca "*tkinter*" que foi usada para criar a interface gráfica do utilizador, o que era um dos requisitos do projeto, o chat foi também construído utilizando as bibliotecas "*socket*" e "*threading*" do Python, permitindo assim que várias pessoas se possam conectar e conversar ao mesmo tempo. Para além disso também foram implementadas outras funcionalidades como ver o histórico de mensagens e ver a hora a que cada uma foi enviada. O chat foi testado em diferentes computadores conectados na mesma rede e apresentou bom desempenho.

2. Servidor

Primeiramente foi necessário importar as bibliotecas, "*socket*" e "*threading*", de maneira a ser possível desenvolver o projeto. De seguida foram declaradas as variáveis globais que serão utilizadas posteriormente nas funções, incluindo a porta que será utilizada pelo servidor

```
1   import socket
2   from threading import Thread
3
4   # IP do servidor
5   SERVER_HOST = "0.0.0.0"
6   SERVER_PORT = 5002 # porta que queremos usar
```

Figura 1 - Import das bibliotecas e variáveis

Class log(): Relativamente à classe log, existem duas funções, a função "*writelog*", tem o objetivo de escrever todas as mensagens enviadas através do chat para um ficheiro de texto, de modo a ser possível guardar todas as mensagens enviadas no chat. Esse registo é também útil para que, com função "*logsend*", seja enviado esse registo de mensagens aos clientes que se conectarem para eles poderem ver as mensagens enviadas anteriormente.

```
9   class log():
10
11      def writelog(self, msg):
12          with open("log.txt", 'a') as log:
13              log.write(msg + "\n\n")
14
15      def logsend(self, cs):
16          with open("log.txt", 'r') as log:
17              for line in log:
18                  cs.send(line.encode())
```

Figura 2 - Class Log()

Class Server():

***__init__*:** Criamos uma função "*__init__*", onde começamos por inicializar o "*SERVER_HOST*", "*SERVER_PORT*".

Inicializamos uma variável do tipo "set" para guardar os endereços dos clientes conectados (*cliente_sockets*), e criamos e preparamos o "*TCP socket*" chamado "*s*", que é o *socket* do servidor, ao qual os clientes se vão conectar.

```
21  class server():
22
23      def __init__(self, SERVER_HOST, SERVER_PORT):
24          self.SERVER_HOST = SERVER_HOST
25          self.SERVER_PORT = SERVER_PORT
26          self.separator_token = "<SEP>"
27          #inicializa todos os sockets dos clientes conectados
28          self.client_sockets = set()
29          # criar TCP socket
30          self.s = socket.socket()
31          # tornar o port reutilizavel
32          self.s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
33          # bind do socket ao endereço
34          self.s.bind((self.SERVER_HOST, self.SERVER_PORT))
35          # Esperar por conexoes
36          self.s.listen(5)
37          print(f"[*] Listening as {self.SERVER_HOST}:{self.SERVER_PORT}")
```

Figura 3 - Class Server()

terminar: esta função pode ser utilizada para fechar todos os *sockets* mas não é atualmente utilizada no nosso programa.

```
def terminar(self):
    # terminar sockets dos clientes
    for cs in self.client_sockets:
        cs.close()
    # fechar socket do servidor
    self.s.close()
```

Figura 4 - Fechar sockets

Connect_to_Client: esta função vai entrar num ciclo permanente de tentativas de conectar a clientes. Quando um cliente se conecta e é aceite pelo *socket* é criado um objecto *log* que vai enviar o registo de mensagens a esse cliente.

De seguida o cliente é adicionado ao *set self.client_sockets* e é iniciado um *thread* com a função *listen_for_client.*

```
62      def connect_client(self):
63
64          while True:
65              #esta sempre a espera de novas conexoes
66              client_socket, client_address = self.s.accept()
67              print(f"[+] {client_address} connected.")
68              #enviar log para o cliente
69              log1 = log()
70              log1.logsend(client_socket)
71              # adiciona novo cliente aos sockets conectados
72              self.client_sockets.add(client_socket)
73              #começa um novo thread que espera pelas mensagens de cada cliente
74              t = Thread(target=self.listen_for_client, args=(client_socket,)) #args é um tupple
75              #torna o thread um thread daemon para que termine se o main thread terminar
76              t.daemon = True
77              # iniciar o thread
78              t.start()
```

Figura 5 – Connect_to_client

Listen_for_client: esta função vai entrar num ciclo permanente de tentativas de receção de mensagem. Quando recebe uma, ela é descodificada, escrita no *log*, e enviada a todos os clientes que estão no *set self.cliente_sockets*. Em caso de erro é apresentada uma mensagem e o cliente é desconectado.

```
def listen_for_client(self, cs):
    """
    função que fica a espera de mensagens vindas do socket cs
    quando mensagem é recebida é enviada a todos os clientes conectados
    """
    while True:
        try:
            #fica a espera de uma mensagem do socket cs
            msg = cs.recv(1024).decode()
        except Exception as e:
            # cliente ja nao esta conectado, remover da lista
            print(f"[!] Error: {e}")
            self.client_sockets.remove(cs)
        else:
            #escrever mensagem no log:
            logfile = log()
            logfile.writelog(msg)
            # iterate iterar pelos sockets conectados e enviar mensagem
```

Figura 6 - Listen_for_client

3. GUI e Cliente

O programa que será utilizado pelos clientes começa por inicializar as mesmas variáveis que o servidor, mas neste caso o *Server_Host* deve ser o IP do servidor.

Class GUI: todo o restante código está contido nesta classe sendo que para iniciar a interface basta criar uma instância desta classe.

__init__ e começar: Quando é criado a instância, é criada uma janela centrada no ecrã do utilizador onde se pode ver uma *label* a pedir que o utilizador insira o seu nome, e uma *entry box* onde o utilizador deve escrever.

É criado também um botão que quando premido irá ativar a função ***começar***. isto permitirá avançar para a próxima janela do programa e também começar a receber mensagens do servidor.

```python
class GUI:
    # constructor method
    def __init__(self):

        # janela do chat, escondida até o deiconify
        self.Window = Tk()
        self.Window.withdraw()

        # janela de login
        self.login = Toplevel()
        # Escolher o titulo
        self.login.title("Login")
        self.login.iconbitmap("icon.ico")
        self.login.resizable(width=False,height=False)
        window_width = 300
        window_height = 200
        self.login.configure(width=window_width,height=window_height)

        # get the screen dimension
        screen_width = self.login.winfo_screenwidth()
        screen_height = self.login.winfo_screenheight()

        # find the center point
        center_x = int(screen_width/2 - window_width / 2)
        center_y = int(screen_height/2 - window_height / 2)

        #colocar no centro do ecra
        self.login.geometry(f'{window_width}x{window_height}+{center_x}+{center_y}')

        # label de instruçao
        self.pf = Label(self.login, text="Por favor insira o seu nome para continuar:", justify=CENTER)
        self.pf.place(relheight=0.15, relx=0.1, rely=0.07)

        # Label do nome
        #self.labelNome = Label(self.login, text="Nome: ")
        #self.labelNome.place(relheight=0.3, relx=0.1, rely=0.1)

        # entry box para o nome
        self.entryNome = Entry(self.login)
        self.entryNome.place(relwidth=0.4, relheight=0.12, relx=0.3, rely=0.2)

        # foco na caixa
        self.entryNome.focus()

        # Botao de ação para avançar
        self.continuar = Button(self.login, text="Continuar",  command=lambda: self.comecar(self.entryNome.get()))
        self.continuar.place(relx=0.4, rely=0.55)

        self.Window.mainloop()
```

Figura 7 - Class GUI

```python
def comecar(self, name):
        self.login.destroy() #destroi a pagina de login e cria a do menu principal(layout)
        self.layout(name)

        #criar um thread que fica a espera de mensagens para este cliente e imprime
        t = Thread(target=self.receber)
        #torna um thread um daemon thread para que termine sempre que o main thread termine
        t.daemon = True
        # começa o thread
        t.start()
```

Figura 8 - Função começar

Layout: Aqui é gerada a janela mais importante do programa. A janela tinha sido previamente criada na função *__init__* e é agora tornada visível, nela existem vários elementos mas os mais importantes são a *TextCons* , *EntryMsg*, e *botaoMsg*.

A *TextCons* é uma caixa de texto onde serão apresentadas todas as mensagens enviadas e recebidas.

EntryMsg é uma *entry box* onde será escrita a mensagem a enviar.

botaoMsg activa a função *BotaoEnviar* que irá tratar de enviar a mensagem para o servidor.

```
# Janela principal do chat
def layout(self, name):

    self.name = name #nome do utilizador

    #janela principal
    self.Window.deiconify() #faz aparecer a janela
    self.Window.title("PyChat")
    self.Window.iconbitmap("icon.ico")
    self.Window.resizable(width=True, height=True)
    window_width = 600
    window_height = 550
    self.Window.configure(width=window_width, height=window_height)
    # get the screen dimension
    screen_width = self.Window.winfo_screenwidth()
    screen_height = self.Window.winfo_screenheight()
    # find the center point
    center_x = int(screen_width/2 - window_width / 2)
    center_y = int(screen_height/2 - window_height / 2)
    #colocar no centro do ecra
    self.Window.geometry(f'{window_width}x{window_height}+{center_x}+{center_y}')

    #nome no topo da janela
    self.labelHead = Label(self.Window, text=self.name, pady=5)
    self.labelHead.place(relwidth=1)

    #janela onde as mensagens vao aparecer
    self.textCons = Text(self.Window, width=20, height=2, padx=5, pady=5)
    self.textCons.place(relheight=0.745, relwidth=1, rely=0.08)

    #label onde estara a caixa de texto
    self.labelFundo = Label(self.Window, height=80)
    self.labelFundo.place(relwidth=1, rely=0.825)

    #caixa de texto para a mensagem
    self.entryMsg = Entry(self.labelFundo)
    self.entryMsg.place(relwidth=0.74, relheight=0.06, rely=0.008, relx=0.011)
    self.entryMsg.focus()
```

Figura 9 - Função Layout

```
# botao de envio
self.botaoMsg = Button(self.labelFundo, text="Enviar", width=20, command=lambda: self.BotaoEnviar(self.entryMsg.get()))
self.botaoMsg.place(relx=0.77, rely=0.008, relheight=0.06, relwidth=0.22)

# scroll bar
scrollbar = Scrollbar(self.textCons)
scrollbar.place(relheight=1, relx=0.974)
scrollbar.config(command=self.textCons.yview)

self.textCons.config(state=DISABLED)
```

Figura 10 - Botão de envio e scrollbar

BotaoEnviar: Esta função é ativada pelo botão de enviar a mensagem. Ela apaga o texto que estava no *entryMsg* e começa um *thread* com a função *enviarMsg*.

enviarMsg: Serve para codificar a mensagem com a hora e nome e enviar para o servidor.

```
# função para iniciar thread de envio de mensagem
def BotaoEnviar(self, msg):
    self.textCons.config(state=DISABLED) #torna nao responsivo a comandos
    self.msg = msg
    self.entryMsg.delete(0, END)
    enviar = Thread(target=self.enviarMsg)
    enviar.start()

# funçao para receber as mensagens
def receber(self):···

# funcao para enviar mensagens
def enviarMsg(self):
    self.textCons.config(state=DISABLED)

    date_now = datetime.now().strftime('%H:%M')
    message = (f"[{date_now}]{self.name}: {self.msg}")
    s.send(message.encode())
```

Figura 11 - Função Botão enviar e enviarMsg

Receber: esta função inicia um ciclo permanente que irá tentar receber mensagens. Quando tem sucesso ela descodifica a mensagem e adiciona a nova mensagem á caixa de texto.

Em caso de erro é apresentada uma mensagem e o cliente é desconectado.

```
# funçao para receber as mensagens
def receber(self):
    while True:
        try:
            message = s.recv(1024).decode()

            # Insere a mensagem no textCons
            self.textCons.config(state=NORMAL) #torna responsivo a comandos
            self.textCons.insert(END, message +"\n\n") #end é o local onde inserir

            self.textCons.config(state=DISABLED)
            self.textCons.see(END) #move o cursor
        except:
            print("Ocorreu um erro!")
            s.close()
            break
```

Figura 12 - Função Receber

4. Diagrama de classes

Aqui apresentamos o diagrama de classes presentes nos programas de servidor e cliente:

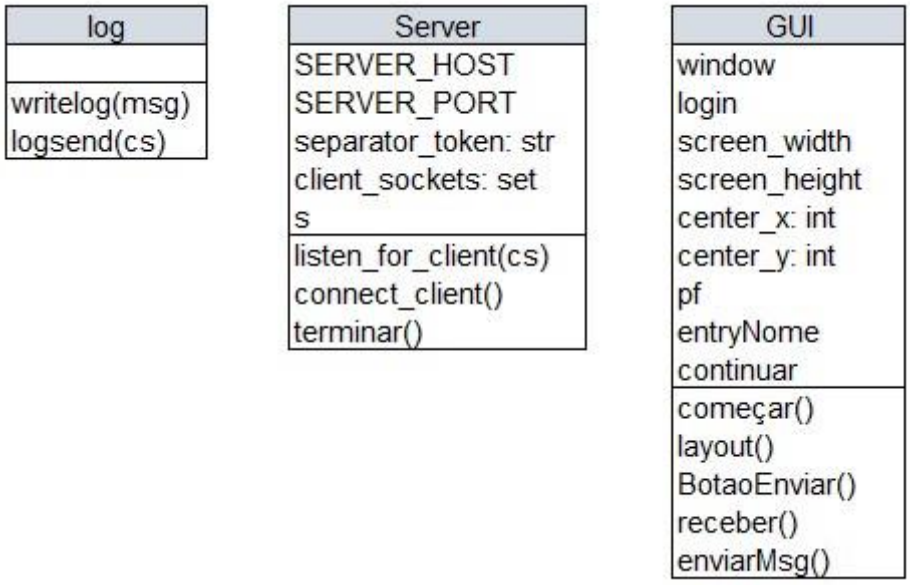

Figura 13 - Diagrama de classes

5. Resultados

Os programas na sua versão final permitem estabelecer um servidor onde outros computadores se podem conectar para enviar mensagens de uma maneira intuitiva e semelhante a outros meios de comunicação modernos.

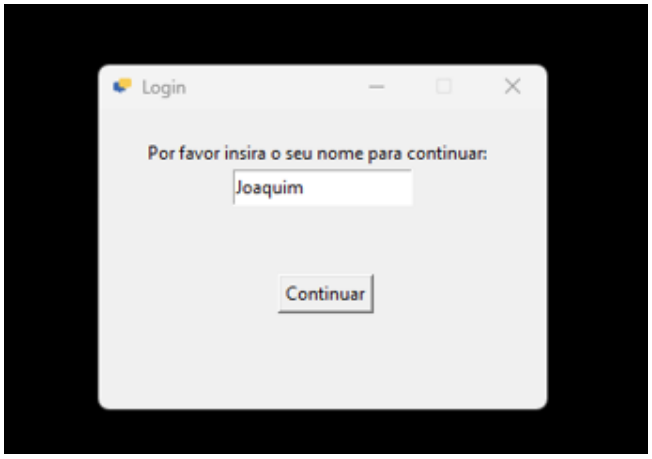

Figura 14 - Página do login

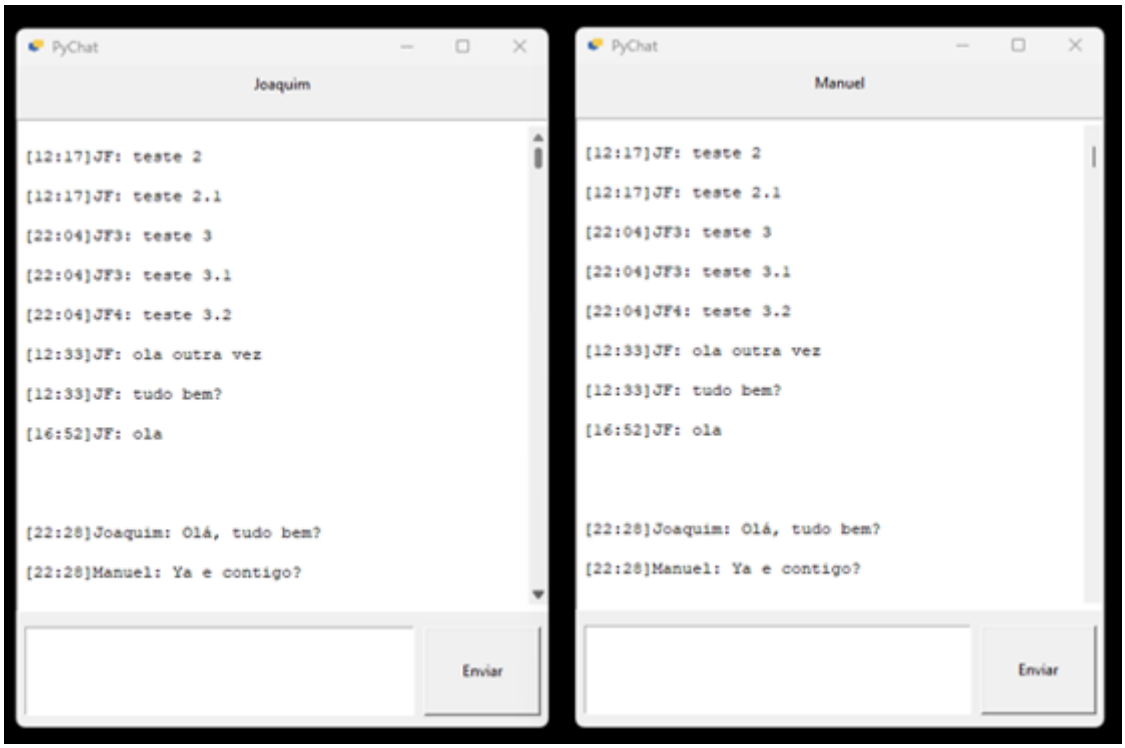

Figura 15 - Teste de conversa