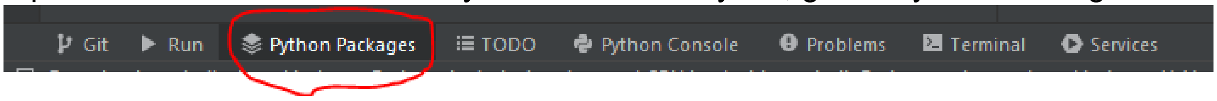
CUNEY Mattéo RT211

User(s) Documentation

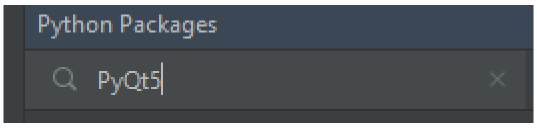
Hello dear users. This document is the official documentation for the users like you who want to know how to install the necessary tools, start them and use the client.

I. How to install the necessary tools

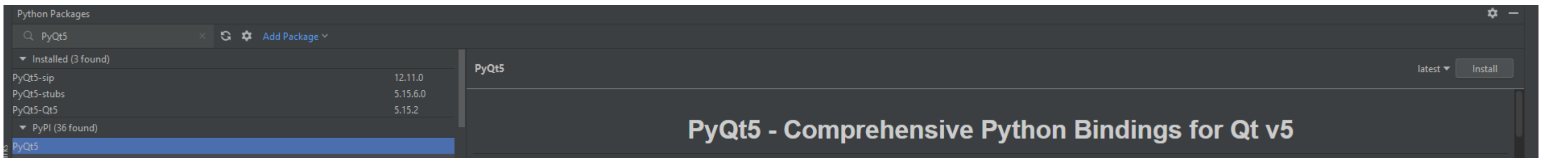
Firstly, you need to install python, my version is python 3.10. Then you need to make a server and a client chat with an GUI interface. For the server, you need to import : « socket, sys, subprocess, platform, psutil ». You also need to write « from ipaddress import IPv4Network » for one request. For the client, you need to import « pathlib ». You need to install PyQt5. For install PyQt5, go on Python Packages



Then write PyQt5 in the research bar.



Then choose PyQt5 and click on install.



Now, you can write :

```
from PyQt5.QtGui import QFont, QKeySequence
from PyQt5.QtWidgets import QApplication, QWidget, QMainWindow, QGridLayout, QLabel, QLineEdit,
QPushButton, QComboBox, \
    QTextEdit, QMessageBox, QFileDialog, QShortcut
from PyQt5.QtCore import QCoreApplication, Qt
from serverAD import *
```

On the server, you need to write at the start of your code :

```
import socket
import sys
import subprocess
import platform
from ipaddress import IPv4Network
import psutil
```

II. How to start them

To start the tools, first of all on the client side. You need to create a "Client" class, then you will set up attributes in a "def __init__(self, your attributes)".

```
class Client():
   def __init__(self, host, port):
      self.__host = host
      self.__port = port
      self.__sock = socket.socket()
      self.__thread = None
```

Then you create a "def connect(self)" to connect to the server with the implementation of an exception that allows you to handle problems connecting to the server. Then the "def dialogue(self)" which will allow the exchange between the client and the server. Then the "def send(self)", which manages the sending of messages to the server while the "def receive(self)", will manage the reception of messages from the server to the client. Finally, the "def connection(self)" sets up the connection.

```
def connect(self) -> int:
   try:
      self.__sock.connect((self.__host, self.__port))
   except ConnectionRefusedError:
      print("[X] Serveur non lancé ou mauvaise information")
      return -1
   except ConnectionError:
      print("[X] Erreur de connection")
      return -1
   else:
      print("[+] Connexion réalisée")
      return 0

def dialogue(self):
   try:
      msg = self.__sock.recv(1024).decode('cp850')
      return msg
   except:
      print("RIP")

def envoi(self,msg):
   self.__sock.send(msg.encode())
   rep = self.__sock.recv(32000).decode()
   return rep

def __reception(self, msg):
   while msg != "kill" and msg != "disconnect" and msg != "reset":
      msg = self.__sock.recv(1024).decode('cp850')
      return msg

def connexion(self):
   self.__sock.connect((self.__host, self.__port))
```

For the GUI part, I created a "MainWindow" class, I set up attributes.

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        widget = QWidget()
        self.setCentralWidget(widget)
        grid = QGridLayout()
        widget.setLayout(grid)
        host = QLabel("Host :")
        port = QLabel("Port :")
```

Then I created all the labels I needed for my GUI. With "self.__the name I want" = Q...... . The Q.... corresponds to the choice of the form that the label will take. It can be a button, a text box, a title or a multiple choice. The "self.__the name I want.hide() or show()" allows you to either hide or show the label.

```
self.__ip = QComboBox()
self.__text2 = QLineEdit("")
self.__cmd = QLabel("Commande :")
self.__text3 = QLineEdit("")
self.__conn = QPushButton("Connexion")
self.__deco = QPushButton("Déconnexion")
self.__send = QPushButton("Envoyer")
```

The "self.shortchut_..." allows me to press enter to send my command, once connected to the GUI. The "grid.addWidget" part allows you to position the various labels you have previously set up on the GUI window as you wish. The first number corresponds to the row and the second number corresponds to the column.

The "self.__....clicked..." links the buttons set up with the "def" set up further down in the code.The "self.__....setText" command is used to display something basic.

Then come my "def". The first one, "def actioncmd" will handle my commands that I would perform in the GUI with an exception for errors.

```
def _actioncmd(self):
    msg = self.__text3.text()
    try:
        rep = self.__client.envoi(msg)
    except:
        self.__lab3.append(f"Erreur\n")
    else:
        self.__lab3.append(f"{rep}\n")
        self.__text3.clear()
```

The def "def actionconn" is the connection part of the GUI to connect to the server from the GUI.

```
def _actionconn(self):
    host=str(self.__ip.currentText())
    port=int(self.__text2.text())
    self.__client = Client(host, port)
    self.__client.connexion()
```

The def "nwclient" is used to create a new client by launching a new GUI window, to have several clients at the same time.

```
def _newclient(self):
    self.__nc = MainWindow()
    self.__nc.show()
```

The def "nvhost" allows you to choose a host (ip address) and add a new one, which will also be added in a .txt file that you select.

```
def _nvhost(self):
    if self.__text5.text() != "":
        ip = self.__text6.text()
        file = open(f"{ip}", "a")
        file.write(f"\n{self.__text5.text()}")
        self.__ip.addItem(self.__text5.text())
        self.__text5.setText("")
        self.__fichier = self.__text6.text()
    else:
        msg = QMessageBox()
        msg.setWindowTitle("Erreur")
        msg.setText("Il est impossible d'ajouter un host vide ! ")
        msg.setIcon(QMessageBox.Critical)
        x = msg.exec_()
```

This will be implemented by the def "nmfichier" which will fetch your .txt file with the different ip addresses written in it. For the last two defs, there are exceptions to handle errors.

```
def _nmfichier(self):
    try:
        settings = QFileDialog.Options()
        settings |= QFileDialog.DontUseNativeDialog
        nom, _ = QFileDialog.getOpenFileName(self, "Choisissez votre fichier", "","Fichiers Texte (*.txt)",
options=settings)
        ip = pathlib.Path(nom).name
        self.__text6.setText(nom)
        file1 = open(f"{ip}", 'r')
        Lines = file1.readlines()

        count = 0
        for line in Lines:
            count += 1
            self.__ip.addItem(line.strip())

        self.__label6.setEnabled(True)
        self.__conn.setEnabled(True)

        self.__text5.show()
        self.__label5.show()
        self.__label6.show()

    except:
        msg = QMessageBox()
        msg.setWindowTitle("Erreur")
        msg.setText(" Vous n'avez choisi de fichier. Veuillez en choisir un ! ")
        msg.setIcon(QMessageBox.Critical)
        x = msg.exec_()
```

Then the def "actiondeco" and "actionquitter" allow you to disconnect and exit the GUI window. Finally, the last part of the code is used to launch the GUI.

```
   def _actiondeco(self):
        self.__client.envoi("disconnect")
        QCoreApplication.exit(0)

    def _actionQuitter(self):
        QCoreApplication.exit(0)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec()
```

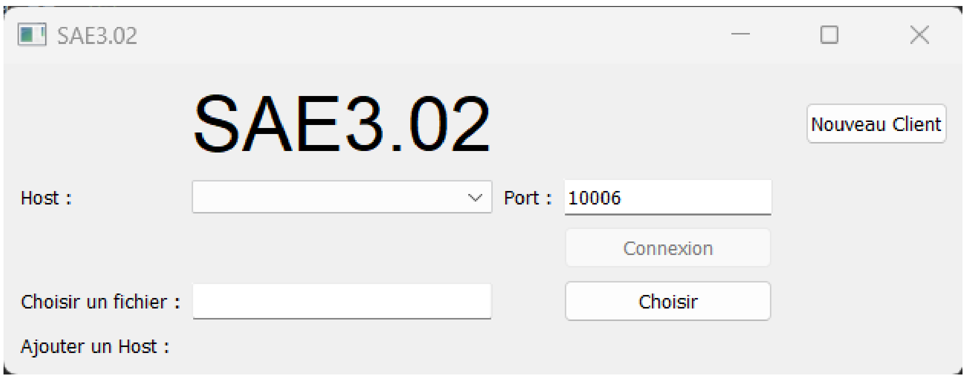
For the server, firstly, I created a "server" def to set up my server. I set up the port and the host with which you can connect to the server from the client. I set up some features such as if the message sent or received = "kill" or "reset" or "disconnect", it performs the action that this word means. I have set up an exception for the connection. Then I set up a loop with my commands that will run as soon as the message sent by the server matches the word I chose.

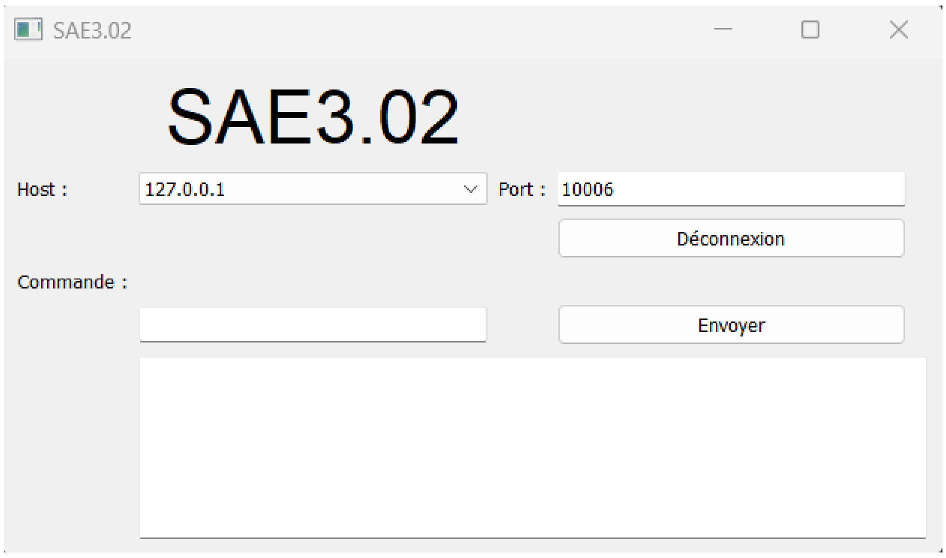
```
if msg == 'OS':
    conn.send(f"{platform.platform()}".encode())

elif msg == 'CPU':
    cpup = psutil.cpu_percent()
    cpu = str(cpup)
    conn.send(f"Utilisation des CPU : {cpu} % .".encode())
```

All this is handled by an exception if there is a problem with the application.

III. How to use the client

Now, to use the client, you must first "run" your server as well as your client. Then you arrive at the first GUI window. First of all you have to fetch or choose your .txt file with the ip addresses written in this .txt file. Once you have chosen your file, you can add a host if you want, or choose with the arrow the one you want and then click on connect. This will bring up a second window with the command section where you can write your commands in the small rectangle, then press enter and you will get the answer in the large rectangle. There is a history to see the results of your previous orders. This works for Windows, Linux and MacOs commands, provided you are on one of these OSes and run the commands for that OS.


SAE3.02
SAE3.02
Nouveau Client
Host :
Port : 10006
Connexion
Choisir un fichier :
Choisir
Ajouter un Host :


SAE3.02
SAE3.02
Host : 127.0.0.1
Port : 10006
Déconnexion
Commande :
Envoyer