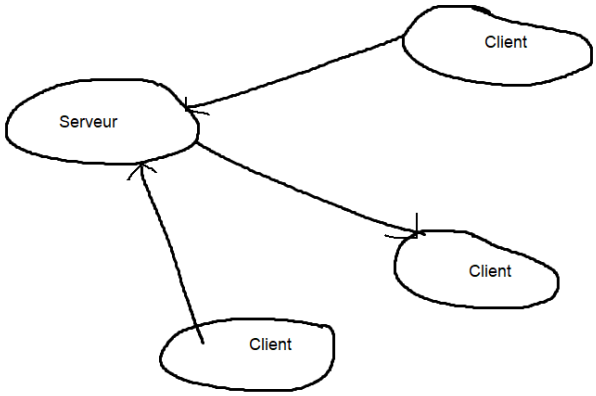
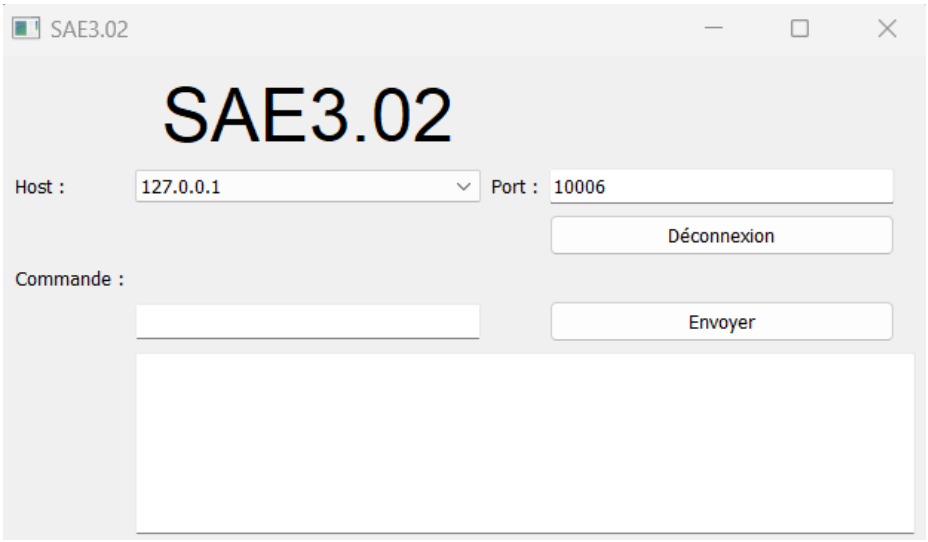
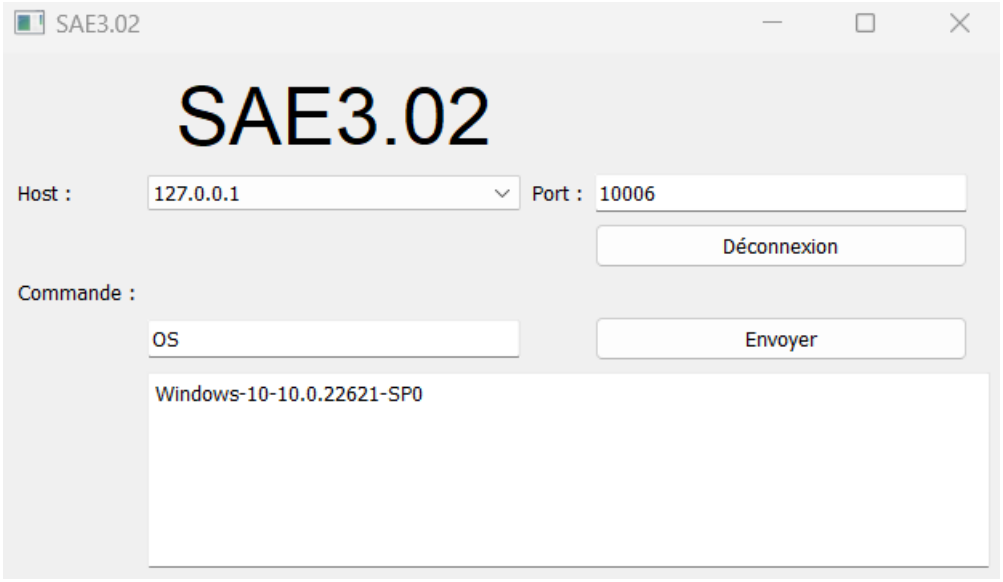
# Programmer Documentation

## I.




First, it works as a client-server chat system. The client communicates with the server and vice versa. A customer's message is relayed to other customers. Each client can resend a message asynchronously. Then concerning the GUI, there is the first window which serves as a login page. We choose our file in.txt format and we can either choose the ip in the file or add them.  You can also create a new client with the bouton at the top-right. After realizing this, you can connect to the server and thus arrive on a second window. This second window is the one where we can enter our commands and have the results of these depending on the OS on which we are.


II.

My code is structured in several def within a class. For the client, I have two class. The first is for the client with inside, 5 def which manages the connection, the sending of message, the reception of message, the dialogue between the client and the server. Then I have a second class. The latter manages all the GUI part and is linked to the client class in order to allow the proper functioning of the GUI and the client to be able to connect to the server from the GUI and to exchange with the latter. My GUI class has 7 def. Each manages an action on the GUI. The first manages the commands that I can enter with an historic of the answer, the second the connection to the server, the third, a new client, the fourth allows when it manages the establishment of new host. The fifth allows you to manage the search and the name of the.txt file you will need to connect. The last two def handle logging out and exiting the GUI.

```
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()
```

```
def _actionconn(self):
    host=str(self.__ip.currentText())
    port=int(self.__text2.text())
    self.__client = Client(host, port)
    self.__client.connexion()
    self.__lab3.show()
    self.__send.show()
    self.__cmd.show()
    self.__text3.show()
    self.__label3.hide()
    self.__label4.hide()
    self.__conn.hide()
    self.__deco.show()
    self.__label5.hide()
    self.__label6.hide()
    self.__text5.hide()
    self.__text6.hide()
    self.__newclient.hide()

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

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_()

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_()

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

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

All this with an "if name" to manage and launch the GUI.

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

III.

For this project, I carried out all the tasks requested. The tasks that I didn't do or didn't do very well are the following. Handling some errors with exceptions because I don't know exceptions very well yet. The kill, reset and disconnect part works but does not close the window. I did not find the solution. A graph of CPU and memory usage. The secure and encrypted connection between client and server. For the last two tasks, I simply didn't have time to do them.