SAE 3.02 clients to server

DIFFICULTY LEVEL SELECTED MEDIUM

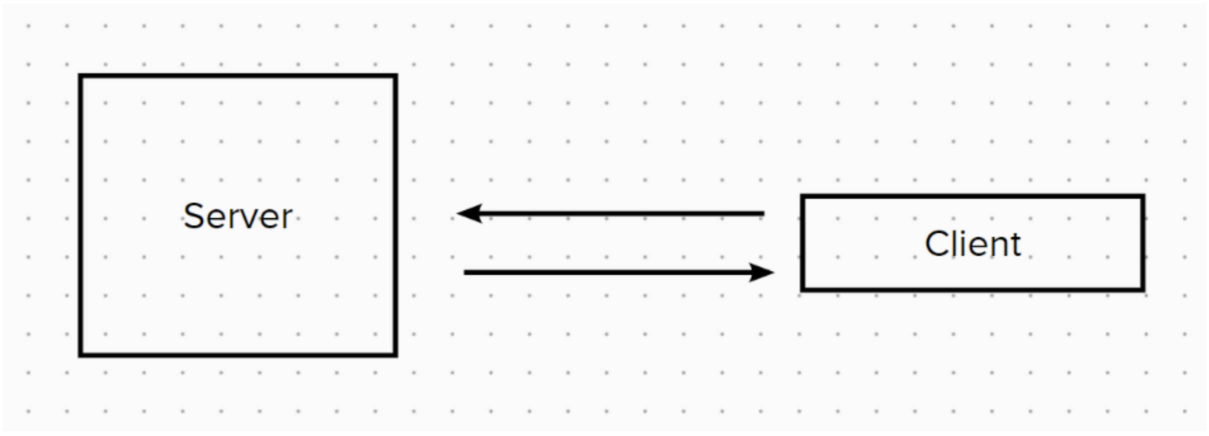
Introduction

In this SAE, we will take a look at the process of building an application from backend to frontend. We will also learn how to use popular Python add-ons such as Pyqt5.

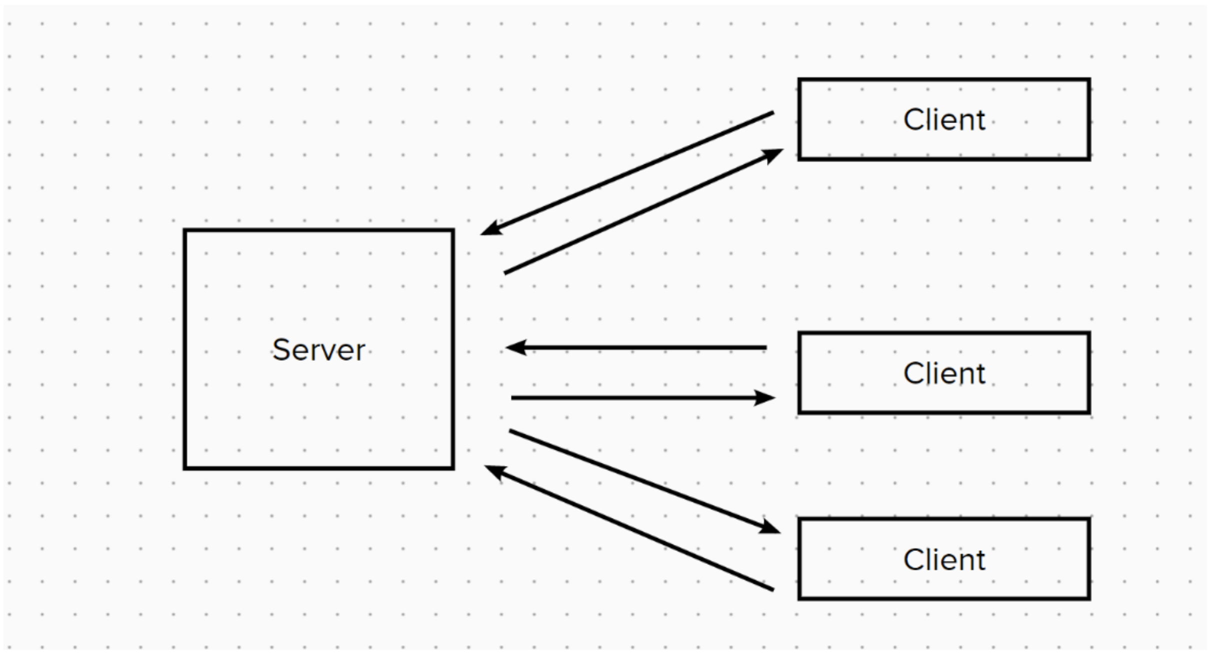
Context

The rules given in this SAE varies in function of the difficulty chosen

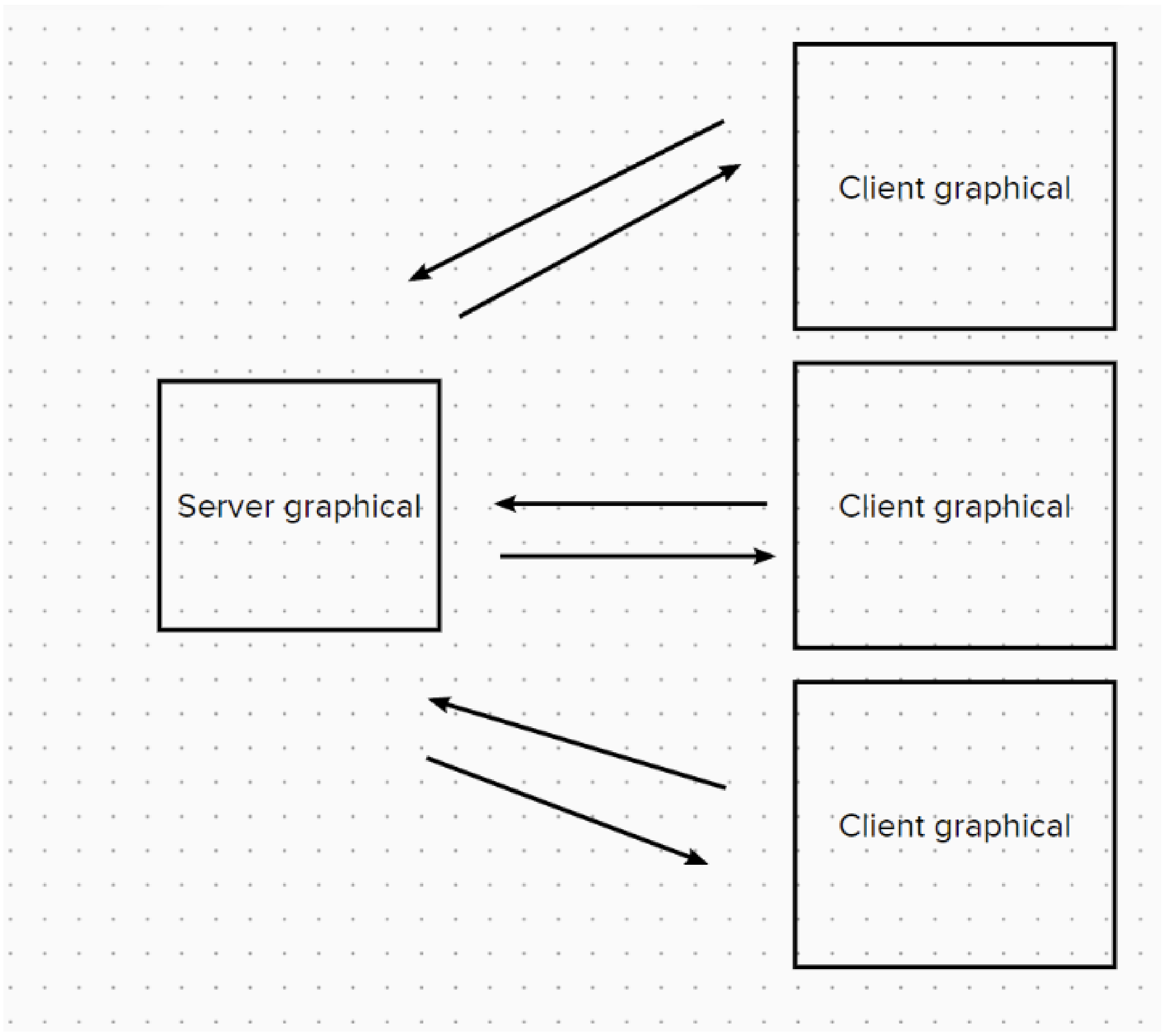
`The easy level would be a simple server-client communication with a chat that allows you to discuss, some commands like "os," "ram," and "kill.”`



`The medium level would be a server-client communication with a graphical interface made with PyQt5. On this interface you would find a "Connect" button, a "Chat" box, and commands just like the easy level.`

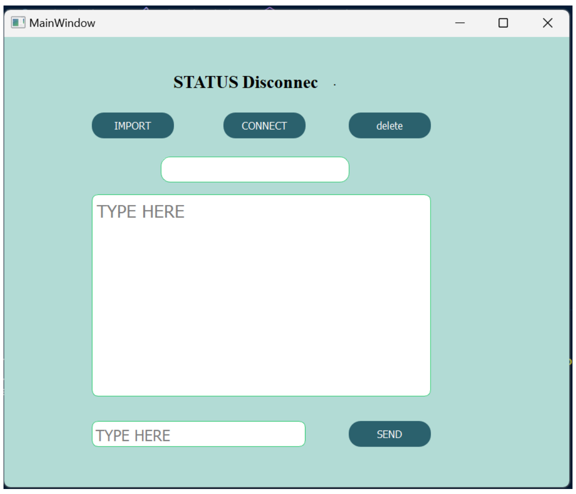


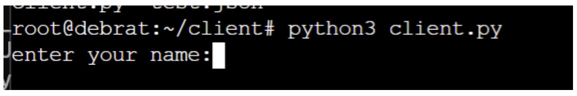
`The hard level would be a server-client communication with a graphical interface on both ends. On the graphical interface, you would be able to choose between the client. You would be able to chat, send commands, and more.`



Building the project

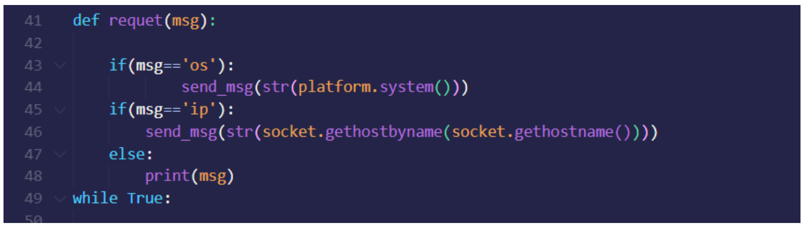
I have made two parts for the server side, front end, and back end. The front end has graphical interface and such. While the back end has functional scripts that bind to a socket.





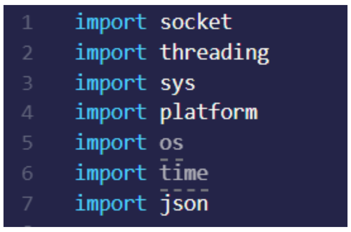
```
root@debrat:~/client# python3 client.py
enter your name:
```

First of all, I chose the medium level because I find it really interesting developing apps on Python with a graphical interface. I'm passionate about the UI frontend, but the backend would be more complicated for me. So, I went on and built the project without a graphical interface and went on to build the commands so that the user could fetch things such as OS, RAM, kill.



```python
def requet(msg):

    if(msg=='os'):
            send_msg(str(platform.system()))
    if(msg=='ip'):
        send_msg(str(socket.gethostbyname(socket.gethostname())))
    else:
        print(msg)
while True:
```

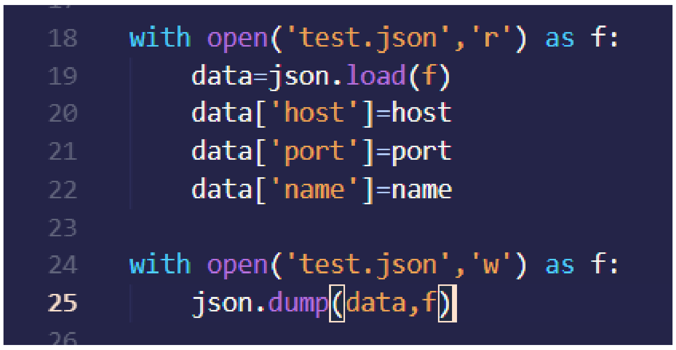
client.py scripts that printed answers back to the server.py



```python
import socket
import threading
import sys
import platform
import os
import time
import json
```

imported modules

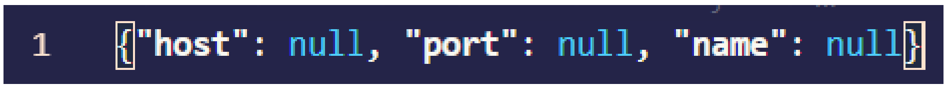
Starting to build the client.py allowed me to have an idea on how to make the server.py.



```python
with open('test.json','r') as f:
    data=json.load(f)
    data['host']=host
    data['port']=port
    data['name']=name

with open('test.json','w') as f:
    json.dump(data,f)
```

dumping data into the test.json

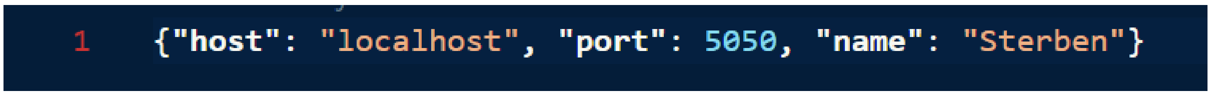
Each time the client.py was launched, I wanted it to generate its address and content into a .json file.



```json
{"host": null, "port": null, "name": null}
```

test.json by default

This generated file will allow the server.py to connect and communicate with the client.py.



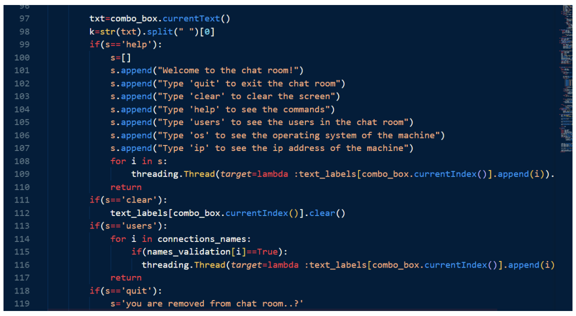
```json
{"host": "localhost", "port": 5050, "name": "Sterben"}
```

test.json after setting up the client

This generated file imported on the server.py would be added.

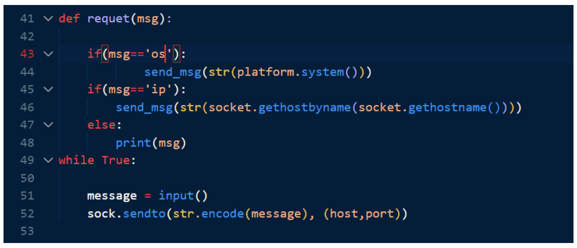
I spent a lot of time working on this project and I hope it will meet your expectations.

It was really something I was passionate about since we had to build a sort of RAT (remote acess trojan) , because it gave us access to the client's machine and let me do anything I wanted to. It was possible to add commands and scripts on the function.py



```python
        txt=combo_box.currentText()
        k=str(txt).split(" ")[0]
        if(s=='help'):
            s=[]
            s.append("Welcome to the chat room!")
            s.append("Type 'quit' to exit the chat room")
            s.append("Type 'clear' to clear the screen")
            s.append("Type 'help' to see the commands")
            s.append("Type 'users' to see the users in the chat room")
            s.append("Type 'os' to see the operating system of the machine")
            s.append("Type 'ip' to see the ip address of the machine")
            for i in s:
                threading.Thread(target=lambda :text_labels[combo_box.currentIndex()].append(i)).
            return
        if(s=='clear'):
            text_labels[combo_box.currentIndex()].clear()
        if(s=='users'):
            for i in connections_names:
                if(names_validation[i]==True):
                  threading.Thread(target=lambda :text_labels[combo_box.currentIndex()].append(i)
            return
        if(s=='quit'):
            s='you are removed from chat room..?'
```

Function.py on the server

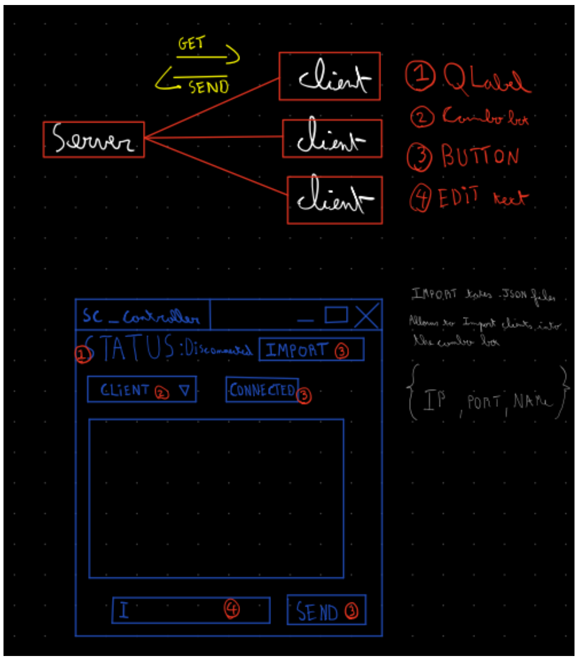


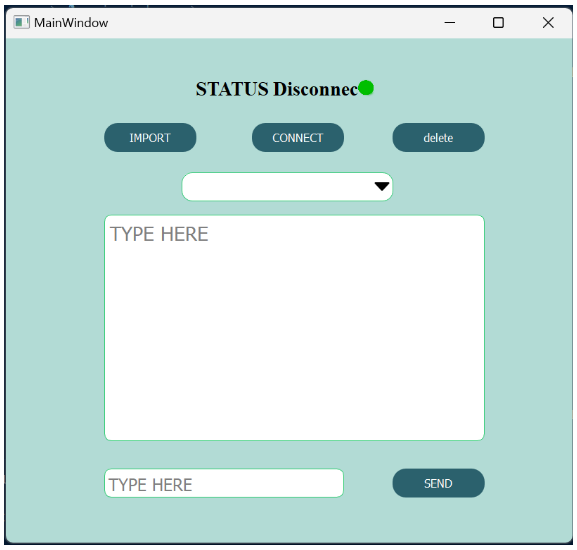
```python
def requet(msg):

    if(msg=='os'):
            send_msg(str(platform.system()))
    if(msg=='ip'):
        send_msg(str(socket.gethostbyname(socket.gethostname())))
    else:
        print(msg)
while True:

    message = input()
    sock.sendto(str.encode(message), (host,port))

```

Client.py which run the command sent and give and print an answer

Any update could be pushed to add more fun scripts.

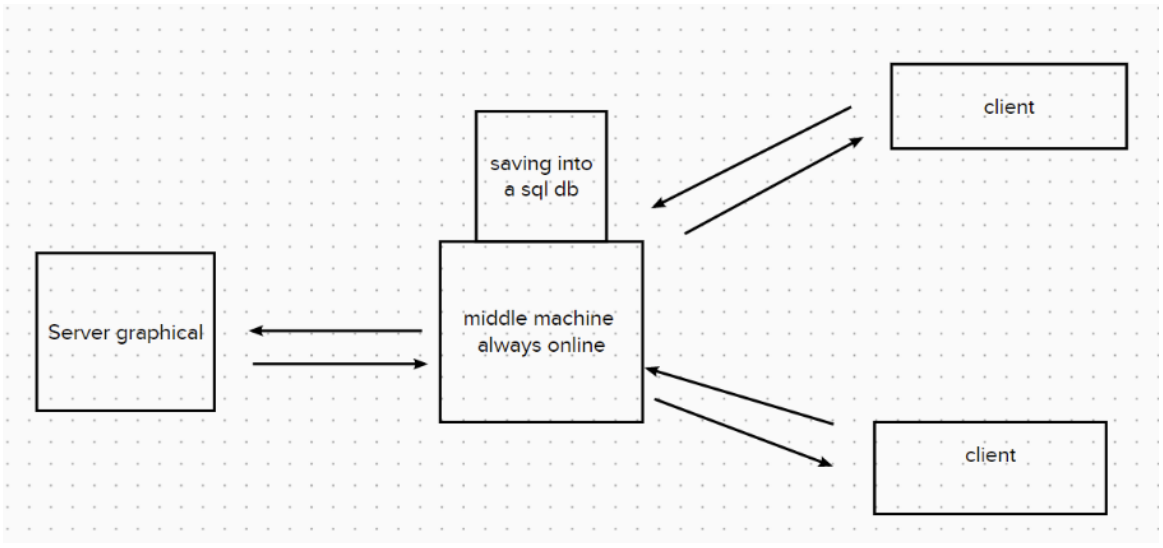
Building the front end was the most fun part of this project because it allowed me to get creative. Making a small canvas to inspire myself and push my creativity was satisfying.





Line 16 to 155 in the main window.py file is dedicated to the front end.

What I could've done to make the project more functional



If we added a third step into the canvas, we could have made a middleman server that could have given us a URL for connection and encryption, enhancing the security between services. Setting up and launching the project would have been way smoother as the server.py could directly search for data directly on the middleman server and control the client.py, the same goes for the client.py, it could send data to the middleman server and directly connect to it instead of waiting for a server.py to be started or launched.

The only problem would've been the time, and the rules. If I weren't as busy on alternating days, I could've gone for this solution, but the rules only indicated a relationship between client and server, not in between. So I tried to give my best to the solution of client and server.

Thank you for reading my documentation. I hope you enjoyed it and wish you a happy new year.