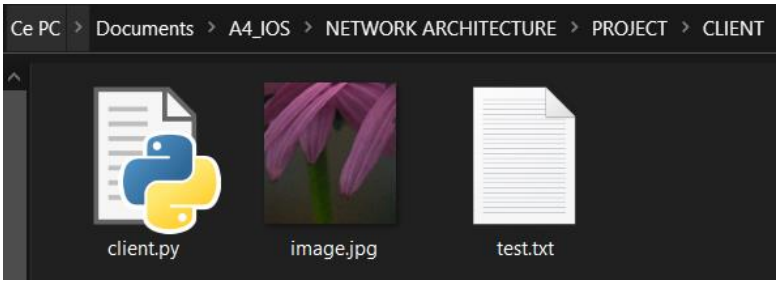
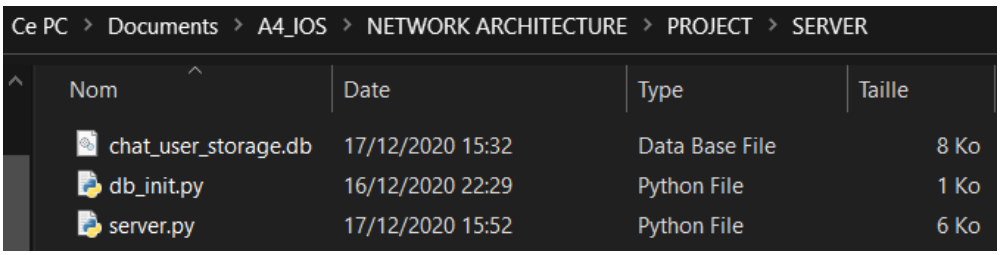
Server Project Chat : Report

Max RENAUDON
Thomas REY
Amel SEDDIK

We decide to code our Network Programming Project in **Python**.

Our initial files are :





First, we need to *launch the db_init.py* file to be sure that the database *chat_user_storage.db* is correctly initialized.

```python
import sqlite3

con = sqlite3.connect('chat_user_storage.db')
cur = con.cursor()
cur.executescript("""
    create table logins(
        name,
        password
    );
    insert into logins(name, password)
    values ('max', '1234');

    insert into logins(name, password)
    values ('thomas', 'pass');

    insert into logins(name, password)
    values ('amel', 'super');

    insert into logins(name, password)
    values ('Michel', 'okay');

    insert into logins(name, password)
    values ('Philippe', 'professeur');

    insert into logins(name, password)
    values ('zizou', '1998');

    """)

con.commit()
con.close()
```

Then we will execute the codes through the command prompt.

1) Connection of Clients to Server in UDP or TCP

Server launch and connections of the clients:

```
C:\Windows\System32\cmd.exe - python server.py
Microsoft Windows [version 10.0.19041.685]
(c) 2020 Microsoft Corporation. Tous droits réservés.

D:\Documents\A4_IOS\NETWORK ARCHITECTURE\PROJECT\SERVER>python server.py
Server is listening...
```

Users had to enter his name, he is next connected to the server with the host and port that we had entered in our code.

4) Multi-threaded more clients (50) per server

With a number in the function listen(), it allows a maximum of 50 users on our server.

```
server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
server.bind((host, port))
server.listen(50)
```

Threads of a client

```
receive_thread = threading.Thread(target=receive)
receive_thread.start()

write_thread = threading.Thread(target=write_msg)
write_thread.start()
```

Threads of the server to handle multiple clients.

```
thread_handle = threading.Thread(target=handle, args=(client,))
thread_handle.start()

thread_kill = threading.Thread(target=kill)
thread_kill.start()
```

6) The server requires a username and password to connect. (use SQLite for password management) the database must be on the server

7) Log log (login, @IP, date, etc.) on the server

Condition : only if he is in the database, then he can join the chat !

On the user interface, he is noticed when he is connected to the server.

```
D:\Documents\A4_IOS\NETWORK ARCHITECTURE\PROJECT\CLIENT>python client.py
Enter your username >not
Enter your password >valid
Username or Password incorrect !
Enter your username >thomas
Enter your password >pass
thomas joined the chat
Connected to the server !
```

On the server, when someone connects to the server, it displays his IP address, the port used and his username.

```
D:\Documents\A4_IOS\NETWORK ARCHITECTURE\PROJECT\SERVER>python server.py
Server is listening...
New client connected with ('192.168.0.29', 61294) at 2020-12-17 15:56:14.758134
Username of the client is amel
New client connected with ('192.168.0.29', 61310) at 2020-12-17 15:57:14.131031
Username of the client is thomas
```

Then, two functions allow the clients to send and receive messages coded in ASCII to and from all other clients connected.

The chat between the clients appears like below. When a client sends a message, it is sent to all other users connected on the server.

AMEL

```
bonjour
amel: bonjour
thomas: salut
comment ca va ?
amel: comment ca va ?
thomas: bien
```

THOMAS

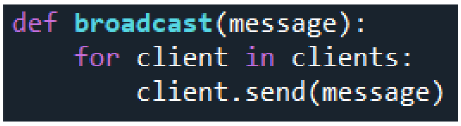
```
amel: bonjour
salut
thomas: salut
amel: comment ca va ?
bien
thomas: bien
```

2) Ability to send and receive messages (chat) (280 characters per message)

All the inputs of the client are managed by the method *write_msg()*

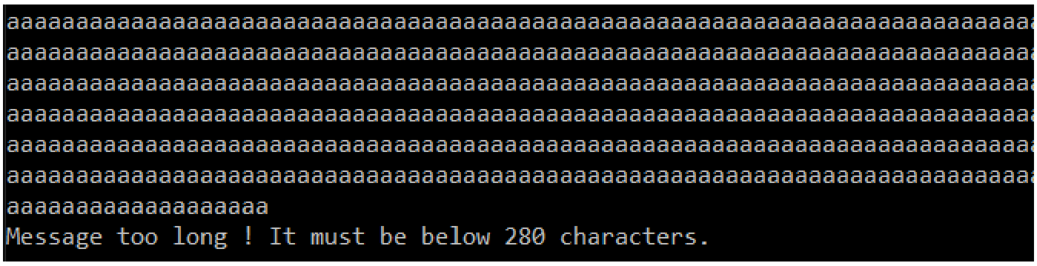
```python
def write_msg(): # Handles the input of the client
    while True:
        client_input = f'{input("")}'
        if(client_input.startswith('#upload')): #upload filename
            upload(client_input.split(' ')[1])
        elif(client_input.startswith('#download')): #download filename
            download(client_input.split(' ')[1])
        else:
            if(len(client_input)<280):
                message = f'{username}: {client_input}'
                client.send(message.encode('ascii'))
            else:
                print('Message too long ! It must be below 280 characters.')
```

The server’s method **broadcast()** sends a message to all the clients



```
def broadcast(message):
    for client in clients:
        client.send(message)
```

We put the limit of a message up to 280 characters, an error message is specified if the message is too long.

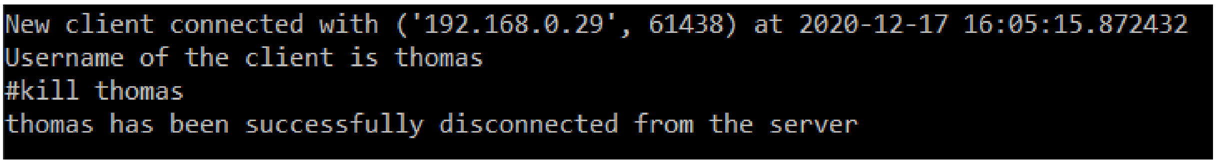


```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaa
Message too long ! It must be below 280 characters.
```

5) Command line on the server (for example #Kill Tim) to disconnect the client Tim

On the server, we can use the command *#kill ‘name of a client’* to disconnect the specified client and warn him and the rest of the clients that he has been disconnected from the server.

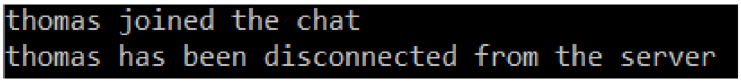
#kill command from the server to disconnect a specific client :



```
New client connected with ('192.168.0.29', 61438) at 2020-12-17 16:05:15.872432
Username of the client is thomas
#kill thomas
thomas has been successfully disconnected from the server
```

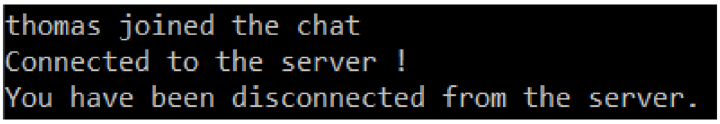
The other clients are warned about the disconnection.

Other clients :



```
thomas joined the chat
thomas has been disconnected from the server
```

Disconnected client :



```
thomas joined the chat
Connected to the server !
You have been disconnected from the server.
```

3) Transfer files (jpeg or others) (client to client, client to server)

- **Uploading a file**

Any client can upload a file on the server to be available for everybody who is connected to download. He must use the command *#upload ‘name of the file’*.

```
#upload test.txt
File sent !
New file test.txt sent by amel is now available from the server
```

```
#upload image.jpg
File sent !
New file image.jpg sent by thomas is now available from the server
```

The server receives the files sent by the clients and can now send them to other clients if they ask.

```
File received test.txt (sent by amel)
File received image.jpg (sent by thomas)
```

Below you can see the state of the SERVER directory after the two previous uploads.

Ce PC > Documents > A4_IOS > NETWORK ARCHITECTURE > PROJECT > SERVER

Nom	Date	Type	Taille
chat_user_storage.db	17/12/2020 15:32	Data Base File	8 Ko
db_init.py	16/12/2020 22:29	Python File	1 Ko
image.jpg	17/12/2020 15:21	Fichier JPG	37 Ko
server.py	17/12/2020 15:52	Python File	6 Ko
test.txt	17/12/2020 15:59	Document texte	1 Ko

We treat the exception with an error message if the name of the file is not valid or does not exist.

```
#upload invalid.bidon
File not found.
```

Here is the method that allows a client to upload a file. We put a limit of 65536 bytes (ko) for the size of the file.

```python
def upload(filename): # Transfers the selected file to the server
    try:
        filesize = os.path.getsize(filename)
        if(filesize < 65536):
            client.send(f'{UPLOAD}{SEPARATOR}{filename}{SEPARATOR}{filesize}{SEPARATOR}{username}'.
            # start sending the file
            file = open(filename, "rb")
            data = file.read(65536)
            client.send(data)
            file.close()
            print("File sent !")
        else:
            print('The file is too heavy for the server (max 65ko)')
    except:
        print('File not found.')
```

The separators allow us to manage the different type of request from the client and to differentiate the data of a texted message from the data of a file that is transferring between a client and the server.

```
SEPARATOR = '<SEPARATOR>'
UPLOAD = '<UPLOAD>'
DOWNLOAD = '<DOWNLOAD>'
```

The server's method ***handle()*** manages the different type of requests from the clients.

```
def handle(client): # Handle the interactions with the client
    while True:
        try:
            message = client.recv(1024)
            if(message.decode('ascii').split(SEPARATOR)[0]==UPLOAD):
                receive_file(client, message.decode('ascii').split(UPLOAD)[1])
            elif(message.decode('ascii').split(SEPARATOR)[0]==DOWNLOAD):
                send_file(client, message.decode('ascii').split(DOWNLOAD)[1])
            else:
                broadcast(message)
        except:
            if(client in clients):
                index = clients.index(client)
                clients.remove(client)
                client.close()
                username = usernames[index]
                broadcast(f'{username} left the chat'.encode('ascii'))
                usernames.remove(username)
            break
```

Finally, he server's method ***receive_file()*** deals with the upcoming data of the file sent by the client.

```
def receive_file(client, info_file): # This method allows the server to receive a file
    filename = info_file.split(SEPARATOR)[1]
    sender = info_file.split(SEPARATOR)[3]
    file = open(filename,'wb')
    data = client.recv(65536)
    file.write(data)
    file.close()
    available_files.append(filename)
    print(f'File received {filename} (sent by {sender})')
    broadcast(f'New file {filename} sent by {sender} is now available from the server'.
```

- **Downloading a file**

Initial state of the CLIENT directory after we deleted manually the files that we sent to the server.

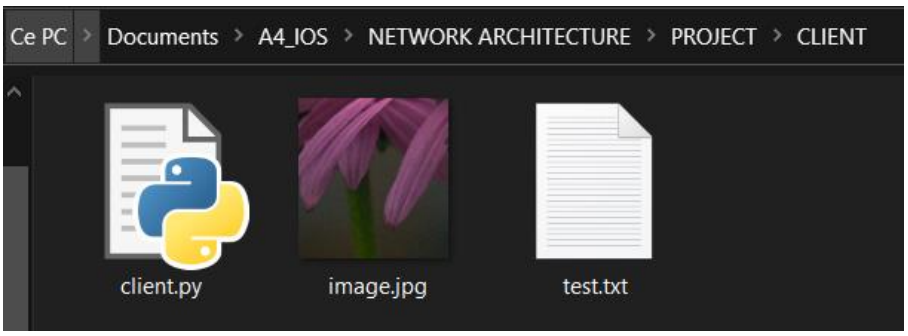
Any client can request for downloading a specific file present on the server (a file that has been previously uploaded by a client).

```
#download test.txt
File test.txt successfully downloaded from the server !
#download image.jpg
File image.jpg successfully downloaded from the server !
```

Response from the server that confirm the upload to the asking client.

```
File test.txt successfully sent to thomas
File image.jpg successfully sent to thomas
```

State of the CLIENT directory after the downloads.



We treat the exception with an error message if the file requested does not exist.

```
#download noexist.ok
Asked file not available on the server :(
```

Here is the method that send the download request to the server.

```
def download(filename): # Asks and receives the asked file from the server
    client.send(f'{DOWNLOAD}{SEPARATOR}{filename}{SEPARATOR}{username}'.encode('ascii'))
```

The server's method ***send_file()*** send the data of the requested file to the asking client.

```
def send_file(client, info_file):
    filename = info_file.split(SEPARATOR)[1]
    receiver = info_file.split(SEPARATOR)[2]
    if(filename in available_files):
        file = open(filename,'rb')
        data = file.read(65536)
        client.send(f'{DOWNLOAD}{filename}'.encode('ascii'))
        client.send(data)
        file.close()
        print(f'File {filename} successfully sent to {receiver}')
    else:
        client.send('Asked file not available on the server :('.encode('ascii'))
```

Then the client's method ***receive()*** manage the different type of response from the server.

```
def receive(): # Handle the message received
    while True:
        try:
            try:
                message = client.recv(1024).decode('ascii')
                if message == 'USER':
                    client.send(username.encode('ascii'))
                elif(len(message.split(DOWNLOAD))==2):
                    filename = message.split(DOWNLOAD)[1]
                    file = open(filename,'wb')
                    data = client.recv(65536)
                    file.write(data)
                    file.close()
                    print(f'File {filename} successfully downloaded from the server !')
                else:
                    print(message)
            except:
                pass
        except:
            print('An error occured !')
            client.close()
            break
```

8) Your original feature

The clients are warned about the disconnection. We treat the exception if a client left the chat or input the wrong command, tell other clients if a client is not connected anymore.

```
thomas joined the chat
thomas has been disconnected from the server
```

```
thomas joined the chat
Connected to the server !
You have been disconnected from the server.
```