# Implementação da fase 1

Nesta fase foi necessário implementar um servidor que controlasse o registo de utilizadores e um Client capaz de comunicar com o servidor via sockets TCP.

**Servidor**

O servidor gere vários clientes, para tal sempre que o servidor aceita uma comunicação TCP é atribuída uma thread , implementada na classe “UserThread”, que gere a receção dos pacotes de um dado cliente.

A classe “Users” mantém a informação de todos os utilizadores registados , para cada utilizador guarda o ip, a porta ,o username, a password e o estado da conecção do utilizador ( se está ativo ou não).

A classe “Server” contém um método para registar utilizadores “registerUser” e outro para apagar utilizadores “logoutUser” que actualizam a informação dos utilizadores através classe “Users”. O método de registo precisa do nome, password, ip, porta e socket atribuídos ao utilizador. O método verifica se o utilizador já se encontra registado, se já estiver as credenciais são verificadas e dá-se o login, se não estiver registado é guardada a informação do utilizador e faz-se o login. Se o registo falhar por já estar registado um utilizador com o mesmo nome é enviado ao cliente uma mensagem a avisar.

**Cliente**

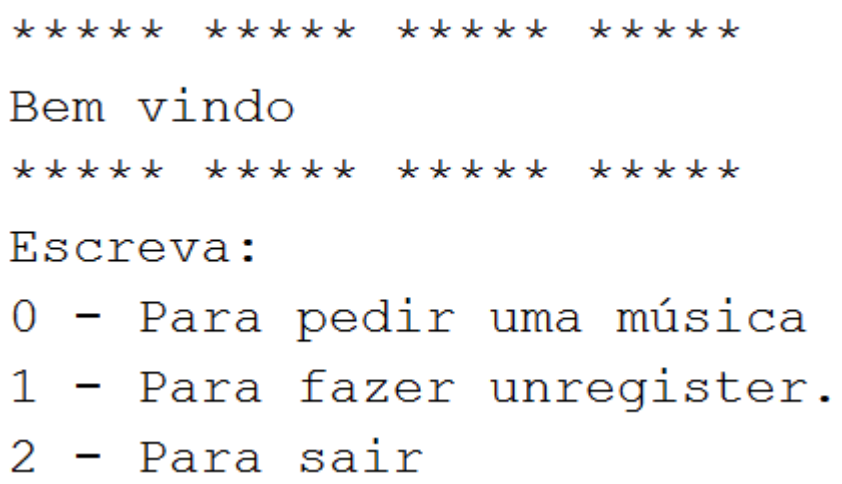
A classe “Client” é o client da aplicação. Para correr o programa são necessários os seguintes argumentos:

./Client -r username password serverPort

O “-r” é para registo de conta e “-l” para apenas login, o “username” e “password” referem-se à conta do utilizador e a “serverPort” à porta que o servidor está à escuta de conecções.

Ao iniciar o Client é  estabelecida a ligação ao servidor “localhost” na porta “serverPort” , de seguida é enviado um pacote de registo para o servidor e iniciada uma thread, classe “ClientReciever”, para gerir os pacotes TCP recebidos do servidor. O Client dispõe também de uma interface para permitir ao utilizador, inserir e pedir uma música e fazer logout da conta.

A seguinte figura é o menu inicial do utilizador.




# Implementação da fase 2

Em primeiro lugar foi finalizada a classe “PDU” que contém todos os métodos para ler e escrever pacotes TCP. A classe visa uniformizar a formatação dos pacotes enviados dentro da aplicação de modo a facilitar e evitar erros na criação e interpretação dos dados em cada pacote.

Os tipos de PDU usados são os do enunciado com a adição do tipo “mensagem” para informar o utilizador com informação de erros do servidor.

Os parâmetros de cada tipo de PDU seguem a informação do enunciado. A formatação de cada tipo é idêntica, contém um header fixo de 3 bytes e é seguido com os campos de cada PDU. Os campos são separados por um “#” para facilitar o parsing.

**Server**

De relevo, no servidor foi adicionado o método “findHosts” que é invocado quando o servidor recebe um pacote “CONSULT_REQUEST”. O servidor envia um “CONSULT_REQUEST” a todos os clientes registados e ativos no domínio do servidor e espera no máximo 500 ms pelas respostas dos outros clientes. O objetivo do método é obter uma lista com a informação de todos os *hosts* que possuíem o ficheiro para depois enviar ao cliente que pediu uma música. A sequência de troca de pacotes TCP entre servidor e clientes num pedido de música é idêntica à do enunciado.

**Optimização UDP**

Esta camada sobre o UDP foi implementada na classe “ClientUDPTransmission” que gere a receção e envio de pacotes que constituem a troca de um ficheiro de música entre 2 clientes.

O datagrama base utiliza 3 bytes do cabeçalho extendido, um byte para tipagem ( Data, Ack, Ret, Fin), outro para sequenciação e um para a *flag more fragments*.

Foi implementado *slow start* e *congestion avoidance* para controlar o tamanho da janela do transmissor. A janela de envio é inicializada a 2 e o *threshold* de passagem para *congestion avoidance* é atingido quando é recebido um *selective reject*. Os pacotes ack não são comulativos e são necessários para aumentar a janela do transmissor.

A retransmissão foi implementada através do envio automático de pacotes *selective reject* . A classe “SendRET” prepara o envio de um *selective reject* com um *timeout* associado. A retransmissão automática é inicializada quando é recebido um pacote de número de sequência superior ao último confirmado, nessa situação são preparadas todas as retransmissões dos pacotes em falta. Ao receber um pacote é cancelado o pedido de retransmissão se existir.

A transmissão é finalizada com o envio de um pacote “FIN” por parte do receptor. Como as confirmações não são cumulativas é provável perderem-se confirmações de alguns pacotes que dificulta a decisão de terminar a ligação no lado do transmissor. Para resolver o problema o recetor quando obtiver o último pacote e tiver todos os pacotes envia um “FIN” que confirma o final da ligação.

# Implementação da fase 3

Nesta fase, foi criado um servidor central com a classe “MusicManager” para gerir o registo de servidores. A este servidor central foi atribuída uma porta fixa para receber conecções TCP de servidores.

Quando um servidor é registado é enviado um pacote de registo para todos os servidores ativos. Esses servidores ao receberem o pacote adicionam o novo servidor à sua lista de servidores ativos. A remoção de servidores dá-se através de um processo idêntico, os servidores em vez de adicionarem à lista de ativos removem.

A classe “Server” foi alterada para receber uma porta como parâmetro. Essa porta é usada para escutar ligações de clientes no domínio do servidor. Foi necessário atribuir uma porta diferente para escutar ligações TCP com outros servidores. Essa porta é escolhida através da soma de 50 unidades ao valor da porta do *serverSocket* de clientes.

A classe “ServerCommunication” gere a comunicação TCP entre os servidores. Esta comunicação é necessária apenas quando um ficheiro de música não existe no

domínio do servidor. Nessa situação o servidor tem de comunicar com os outros servidores, presentes na lista de servidores ativos actualizada pelo servidor central, para obter informação de possíveis *hosts* que possuem o ficheiro de música.

# Testes realizados

Nesta secção apresentámos um exemplo de um teste com 1 servidor central, 2 servidores e 3 clientes. Um cliente para o servidor 2 e os outros 2 para o servidor 1.

O teste consiste em o cliente do servidor 2 pedir um ficheiro de música que está guardado na máquina de um cliente num diferente servidor.

Servidor central regista 2 servidores novos e atualiza as listas de servidores em cada servidor.

```
Conecçao socket aceite
Added Server 0 actualised other servers list
Updated new server 0 active list
Conecçao socket aceite
Added Server 1 actualised other servers list
Updated new server 1 active list
```

N

Servidor2. Regista o utilizador do servidor 2, recebe um consult_request do cliente e procura por hosts fora do servidor. É encontrado um host com o ficheiro pedido. Por último, é enviada a informação do *host* encontrado ao cliente que pediu a música.

```
Servidor online
Recieved Server REGISTER
Added server 0 to active list
Conecçao socket aceite
InputStream lida
PDU type 2
REGPDU read type:1 from:joaoS2
Checking if registed...
User 'joaoS2' registou e ligou
InputStream lida
PDU type 4
Reading CONSULT_REQUEST
CONSULT_REQUEST read banda:banda,filename: 000001.mp3 from:joaoS2
Probing hosts for file
Sending consultRequest to other 1 servers
Connecting to server port 9141
Recieved Server CONSULT_RESPONSE
1 hosts found
Found 1 hosts
Sent ConsultResponse with hosts data to client
```

Servidor 1. Para além de ter gerido o registo dos 2 utilizadores, enviou consult_requests aos utilizadores do seu domínio para encontrar o ficheiro de música.

```
Conecçao entre Servidores aceite
Recieved Server CONSULT_REQUEST
InputStream lida
PDU type 5
User: Tiago responded CONSUT_REQUEST
InputStream lida
PDU type 5
User: Josue responded CONSUT_REQUEST
```

Cliente com o ficheiro.

```
Recieved CONSULLT_REQUEST: asking for file 000001.mp3
I have the file:000001.mp3
Sent CONSULTRESPONSE
Probed by client ip:127.0.0.1on port:55767
File size 2411752 bytes
Number of fragments to send:50
Max data on fragment:49146
LastFragment size:3598
Sent inital 2 packets
Recieved ACK for packet 0
Sent packet 2
Sent packet 3
Recieved ACK for packet 1
Sent packet 4
Sent packet 5
```

Cliente que pediu o ficheiro.

```
Recieved CONSULT_RESPONSE with 1 hosts
Probing hosts to find the lowest OWD
Probing user Tiago with ip:127.0.0.1 on port:63213
ResponseTime: 1465770679191
DatagramPacket successfull load
Client probed, OWD =0
Sending Request to start file 000001.mp3 transfer with user addrss: /127.0.0.1
Writed 1 segments
Sending selective ACK packet 0
Confirmed 1 packets
Sending selective ACK packet 1
Confirmed 2 packets
Writed 1 segments
Sending selective ACK packet 2
Sent RET 3
Confirmed 3 packets
```