

# Computação Distribuída – Guião 1

Diogo Gomes, Nuno Lau, João Rodrigues  
Março 2022

## Introdução

O objectivo deste guião é desenvolver um sistema de chat básico. Os clientes ligam-se ao servidor através de uma socket TCP e enviam mensagens em formato JSON. O servidor recebe as mensagens dos clientes e propaga as mensagens para os restantes clientes ativos. Na Figura 1 tem um esquema simplificado do sistema.

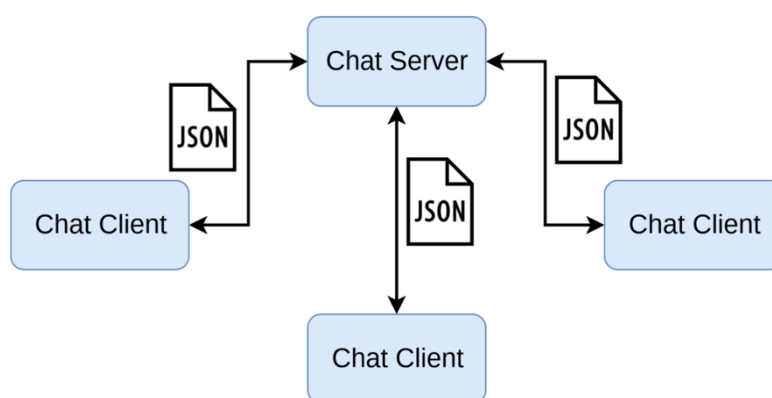


Figura 1 - Esquema simplificado do sistema de chat

## Objectivos:

1. Desenvolver um sistema de chat simples (mais de um cliente).
2. Comunicação através de sockets TCP.
3. Primeiro contacto com GIT e com a plataforma GitHub

## Prazo:

23 de Março de 2022

Entrega através do Github Classroom (automática)

## GitHub Classroom

- Para resolver este guião deverá começar por aceitar o mesmo em <https://classroom.github.com/a/pzhSseKj>
- Ao aceitar o guião será criado um repositório online a partir do qual deve fazer um clone local (no seu computador).
- Deverá enviar as suas alterações periodicamente para o repositório e manter-se atento ao canal #cd em <https://detiuaveiro.slack.com>
- Antes do prazo, o seu código deverá passar os testes automáticos (tab Actions no seu repositório github)

## Requisitos da solução:

1. As mensagens devem obedecer ao formato JSON (ver Protocol).

2. Os programas cliente devem esperar por input do utilizador (que será enviado ao servidor) e por mensagens do servidor.
3. O servidor espera por mensagens dos clientes e propaga para todos os clientes ativos no canal.
4. **Importante:** um cliente não pode bloquear à espera de mensagens ou input do utilizador.
5. **Importante:** o servidor não deve enviar mensagens para clientes que não estão ativos ou num canal diferente
6. O utilizador pode enviar 2 comandos que não são tratados como mensagens:
  - a. `/join <nome canal>` para sinalizar que pretende mudar de canal.
  - b. `exit` para sinalizar que pretende terminar o programa cliente
7. Não pode recorrer a bibliotecas externas que não façam parte da *standard library* do Python 3.7
8. Não pode alterar os conteúdos da pasta **tests**
9. São fornecidos esqueletos de código na pasta **src**, deve utilizar estes ficheiros completando com o código necessário para completar este guião.
10. Todas as mensagens recebidas/enviadas pelos clientes/servidores devem ser escritas para um ficheiro de log (já configurado no código fornecido). Exemplo:  
[logging.debug\('received "%s", message\)](#)

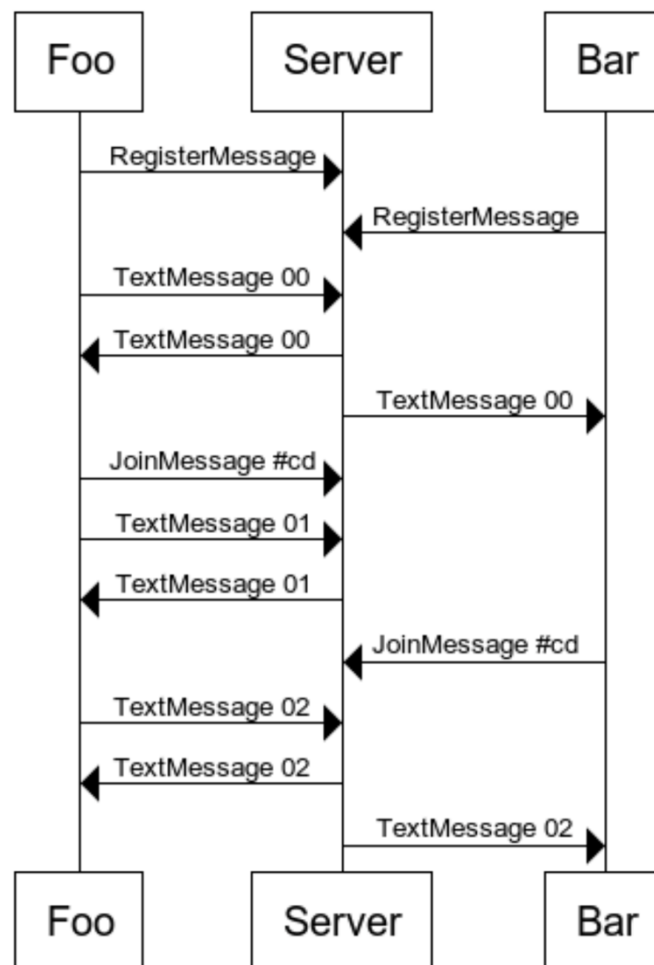


Figura 2 - Diagrama de Sequência de Mensagens

## Exemplo

Na Figura 2 encontra um diagrama de sequência de mensagens que explicita a troca de mensagens que deve ocorrer entre dois clientes (**Foo** e **Bar**) e o Servidor (**Server**).

Cada cliente inicia a comunicação com o servidor enviando uma mensagem do tipo *RegisterMessage* que informa o servidor do seu *username*.

De seguida trocam mensagens entre si, sendo que cada cliente recebe uma cópia da mensagem enviada ao servidor.

O cliente **Foo** envia uma mensagem *JoinMessage* dando conta ao servidor que agora ouve mensagens do canal “#cd”. Como o cliente **Bar** não pertence a este canal, não recebe a mensagem 01.

Posteriormente **Bar** envia uma *JoinMessage* e torna-se também ele membro do canal “#cd” passando a receber mensagens futuras.

## Protocolo:

O protocolo utilizado entre clientes e servidor é baseado na troca de documentos JSON, precedidos do tamanho do documento JSON que lhe segue. O tamanho é um inteiro codificado em 2 bytes em *big endian*. Exemplo de duas mensagens enviadas de seguida:

```
\0%{"command": "nome", "campo": "valor"}\0%{"command": "nome", "campo": "valor"}
```

`\0%` é o tamanho da mensagem em ASCII (\0 representa **um** byte com o valor binário 0; % é o carater ASCII com o valor decimal 37)

Definiram-se um conjunto de classes em `protocol.py` que devem implementar este protocolo.

Class	Representação JSON
<b>RegisterMessage</b>	<code>{"command": "register", "user": "student"}</code>
<b>JoinMessage</b>	<code>{"command": "join", "channel": "#cd"}</code>
<b>TextMessage</b>	<code>{"command": "message", "message": "Hello World", "channel": "#cd", "ts": 1615852800}</code>

A mensagem *RegisterMessage* contém o nome do cliente e necessita ser a 1ª mensagem enviada pelo cliente ao servidor.

A mensagem *JoinMessage* pode ser enviada a qualquer instante.

A mensagem *TextMessage* contém para lá da mensagem propriamente dita um campo “ts” que significa *timestamp*, um *timestamp* não é mais que o número de segundos passados desde 1 de Janeiro de 1970 (UNIX *timestamp*)

## Referências

1. [Python. json – json encoder and decoder](#), 2021.
2. [Python. selectors – high-level i/o multiplexing](#), 2021.

3. [Python. Socket programming howto](#), 2021
4. [Python built-in Types - Additional Methods on Integer Types](#), 2021.
5. [Exemplo de como ler texto do teclado sem bloquear](#), 2021