

**Distributed Object Detection**

**Universidade de Aveiro**

**Licenciatura em Engenharia Informática**

**UC 40382 - Computação Distribuída**

**Docentes:**

Prof. Diogo Gomes

Prof. Nuno Lau

**Trabalho realizado por:**

Eduardo Santos - 93107

Pedro Bastos - 93150

**Bibliotecas utilizadas**

Tanto o server como os workers utilizam ***Flask***, uma framework de **Python** utilizada em Web. É utilizado um **ambiente virtual** para trabalhar localmente. Para os pedidos entre o server e o worker, é utilizada a biblioteca ***Python Requests****,* que permite a interação entre os servidores **HTTP**. Finalmente, para poder enviar vários frames ao mesmo tempo para vários workers, utilizamos a biblioteca ***Python Asyncio*** que, além de simples de usar, é compatível com ***Flask*** e ***Python Requests***.

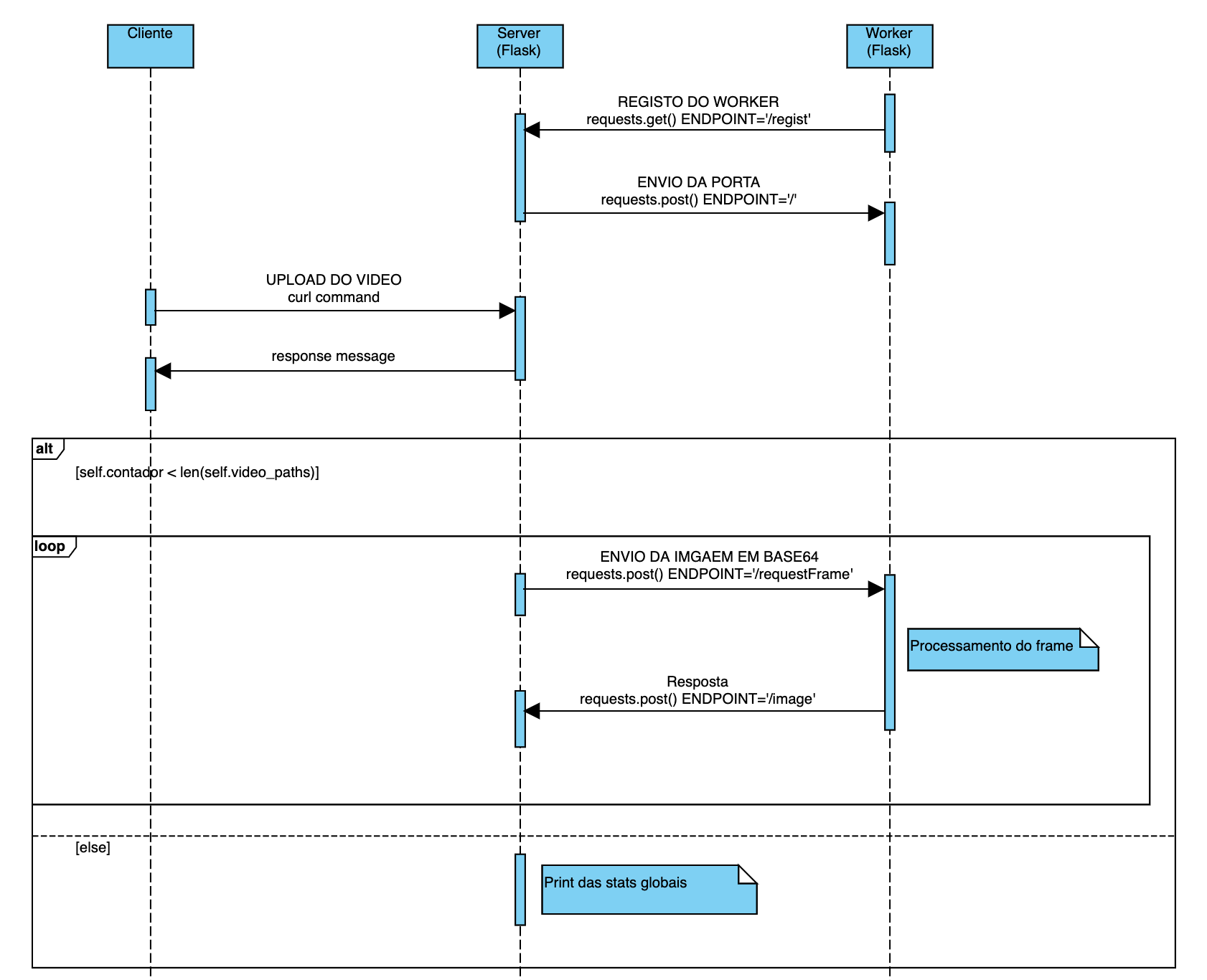
**Protocolo**

Primeiramente, é necessário iniciar o server, este que inicia o ***Flask***e fica bloqueado à espera de pedidos. Em seguida, devem ser iniciados os workers pretendidos. Cada worker, depois de serem iniciadas as bibliotecas do ***tensorflow****,* ***Keras*** *e* ***YOLOv3***por uma questão de desempenho no processamento das frames, faz um pedido get ao URL do server. O servidor responde com a porta designada para esse worker. Depois do worker receber a porta, inicia o ***Flask*** com a mesma.

Seguidamente é feito o upload do vídeo do cliente para o servidor (“curl -F 'video=@moliceiro.m4v' <http://localhost:5000>”). O servidor recebe o video e responde com “Video delivered.”, não obrigando ao cliente esperar pelo processamento do video.

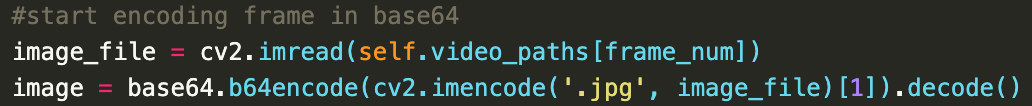
Todas as mensagens entre o server e os workers foram enviadas em formato JSON.

Depois de ler os frames todos do video e guardá-los numa lista, o servidor vai enviar **n** frames fazendo requests.post() de **n** em **n, sendo n o número de workers,** e espera pela resposta. Depois da resposta, verifica se o número de pessoas ultrapassa o definido e emite um alerta, caso existam razões para tal. Seguidamente, verifica se já estão todos os frames processados. Se não estiverem, envia os próximos **n** frames, se estiverem, é chamada a função ***endProcessing()*** que imprime na consola as estatísticas finais do vídeo.

Message sequence chart do protocolo

**Encoding**

Como o objetivo não era enviar os paths dos ficheiros diretamente para os workers, decidimos fazer o encoding dos mesmos utilizando Base64, não só porque o JSON não suporta bytes, mas também de modo a tornar este envio mais eficiente.

****

Encoding

**A picture containing clock

Description automatically generated**

Decoding

**Resultados**

Para a apresentação dos resultados, são impressos alertas no server quando, para cada frame, o número de pessoas presentes no mesmo ultrapassa o número definido inicialmente pelo utilizador. No mesmo alerta podemos ver o número do frame, o número de pessoas detetadas e a classe das mesmas (<person>):



Imagem: Exemplo de alerta

1 Worker:

A screenshot of a cell phone

Description automatically generated

2 Workers:

A screenshot of a cell phone

Description automatically generated

4 Workers:

A screenshot of a cell phone

Description automatically generated

**Observações**

Ao correr o script **run.sh**, o processo do server é overrided pelos processos dos workers, e estes são overrided pelo processo do curl, o que faz com que, no fim, o processo morra sem executar qualquer tipo de processamento dos frames.

Assim, apesar de o programa funcionar correndo normalmente, não funciona com o script.

**Referências**

Slides da UC

[**https://www.geeksforgeeks.org/get-post-requests-using-python/**](https://www.geeksforgeeks.org/get-post-requests-using-python/)

[**https://docs.python.org/3/**](https://docs.python.org/3/)

[**https://requests.readthedocs.io/en/master/**](https://requests.readthedocs.io/en/master/)

[**https://docs.python.org/3/library/asyncio.html**](https://docs.python.org/3/library/asyncio.html)