



**GABRIELA MARCULINO DA SILVA  
LINCOLN AMORIM**

# **SOCKETS**

Documentação do trabalho da matéria de Redes de Computadores

**Profº: Rubens Barbosa Filho**

**DOURADOS  
2022**

**Gabriela Marculino da Silva**  
**Lincoln Amorim**

## **SOCKETS**

Documento para detalhar o processo de desenvolvimento do trabalho de Sockets redigido para a matéria Redes de Computadores pela Universidade Estadual do Mato Grosso do Sul; requerido pelo Profº Rubens Barbosa Filho.

**DOURADOS**

# **SUMÁRIO**

<b>1.SUMÁRIO DO PROBLEMA A SER TRATADO</b>	<b>4</b>
<b>2.ALGORITMOS E TIPOS ABSTRATOS DE DADOS,PRINCIPAIS FUNÇÕES, PROCEDIMENTOS E DECISÕES DE IMPLEMENTAÇÃO.</b>	<b>5</b>
<b>3.DECISÕES DE IMPLEMENTAÇÕES QUE PORVENTURA ESTEJAM OMISSOS NA ESPECIFICAÇÃO.</b>	<b>11</b>
<b>4.COMO FOI TRATADA A RETRANSMISSÃO DE MENSAGEM.</b>	<b>12</b>
<b>5.TESTES MOSTRANDO QUE O PROGRAMA ESTÁ FUNCIONANDO DE ACORDO COM AS ESPECIFICAÇÕES SEGUIDOS DA ANÁLISE.</b>	<b>14</b>
<b>6.CONCLUSÕES E REFERÊNCIAS BIBLIOGRÁFICAS.</b>	<b>19</b>

## **1. PROBLEMA A SER TRATADO**

O intuito deste trabalho era realizar a construção de um proxy funcional para que nos ajudasse a entender de forma linear como devemos realizar uma aplicação deste porte e de como o sistema se comporta ao receber tais dados e de como devemos tratá-los a partir de observações do comportamento do algoritmo sobre a camada de aplicação.

Para concluirmos o objetivo proposto, decidimos utilizar a linguagem Python, mais precisamente a versão 3.10.5. O motivo que nos levou a escolher essa linguagem foi pela sua simplicidade e objetividade, pois continha todas as bibliotecas necessárias e também porque gostaríamos pessoalmente de aprimorar nosso conhecimento nesta tecnologia.

Além de termos escolhido esta linguagem, também escolhemos utilizar threads como método de aceitar múltiplas requisições pois era o método mais eficiente que conhecemos para deixar nosso código mais fluido e funcional.

## 2. ALGORITMOS E TIPOS ABSTRATOS DE DADOS, PRINCIPAIS FUNÇÕES, PROCEDIMENTOS E DECISÕES DE IMPLEMENTAÇÃO.

Como base de construção do servidor, utilizamos threads como base de dados pois achamos que seria uma forma simples e eficiente de fazer o tratamento de recebimento de vários clientes simultaneamente em um único servidor.

Para fazermos isso utilizamos a própria API do Python (`_thread`), que funciona da mesma forma que a API da linguagem C porém de uma maneira mais simples e facilitada tal qual uma linguagem de alto nível.

```
281 while True:
282     # abre a conexao com o cliente
283     c, addr = s.accept()
284     msg = 'Got connection from ' + str(addr)
285     logging.info(msg)
286     _thread.start_new_thread(controlt, (c,))
```

## 2.1 CLIENTE

Para a construção do cliente, utilizamos o IP\_NET 4 para o envio de requisições juntamente com a conexão do socket. A conexão funciona a partir do momento que escolhemos uma porta do qual também será utilizada no servidor e um endereço de IP, como desenvolvemos este trabalho em uma máquina virtual tivemos que tomar um cuidado a mais pois às vezes o endereço de IP resetava e nos causava pequenas confusões.

```
1  # 172.27.3.200 ip virtualmachine
2
3  # Import socket module
4  import socket
5  import sys
6
7  BUFLen=8192
8
9  # Create a socket object
10 s = socket.socket()
11
12 # Define the port on which you want to connect
13 port = int(sys.argv[1])
14
15 # connect to the server on local computer
16 s.connect(('', port))
17
18 # send the url to access and receive the message
19 url = input("Request: ")
20 s.send(url.encode())
21
22 # receive data from the server and decoding to get the string.
23 print (s.recv(BUFLen).decode('utf-8'))
24
25 print (s.recv(BUFLen).decode())
26 # close the connection
27 s.close()
```

## 2.2 SERVIDOR

Para o desenvolvimento do servidor, decidimos fazer de maneira estrutural e global, ou seja, sem a necessidade de fazermos dentro de uma função “main”. Decidimos fazer dessa forma pois achamos interessante esta liberdade que a linguagem Python nos dá e resolvermos usufruir como tal.

Nesta etapa construímos nosso socket vinculada a porta que escolhemos (7777), com isso deixamos o servidor pronto para receber qualquer requisição e pronto para se conectar com o cliente utilizando o gerenciamento de threads para que possa ser tratado de maneira adequada.

```
264 # cria o socket
265 s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
266 s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
267 try:
268     s.bind(('', port, 0, 0))
269 except socket.error:
270     print("Erro no bind da porta:", port)
271     sys.exit(-1)
272 print("Port definido como:", port)
273
274 # coloca o socket em listen
275 s.listen(1)
276 print("Socket is listening")
277 numpedidos = 0
278 numhits = 0
279 numfails = 0
280 # loop
281 while True:
282     # abre a conexao com o cliente
283     c, addr = s.accept()
284     msg = 'Got connection from ' + str(addr)
285     logging.info(msg)
286     thread.start_new_thread(control, (c,))
```

Após a conexão ser realizada de forma correta, em seguida ocorre o tratamento da requisição que seria uma espécie de “filtro” para termos específicos como: GET ou ADMIN. Se for uma requisição válida, o código executa o que foi pedido de acordo com o termo da requisição, caso contrário irá ocorrer o fechamento imediato de conexão do cliente.

```

167     if "ADMIN" in reqsplit:
168         reqsplit[1] = reqsplit[1].upper()
169     numpedidos += 1
170     # trata o request
171     match reqsplit[0]:
172         case "GET":
173             if "if-modified-since" in reqsplit:
174                 try:
175                     c.send(caching.expired(reqsplit[1]))
176                 except:
177                     c.send(b'404 Not Found')
178             else:
179                 try:
180                     c.send(testcache(reqsplit[0], reqsplit[1]))
181                 except:
182                     c.send(b'404 Not Found')
183         case "ADMIN":
184             match reqsplit[1]:
185                 case "FLUSH":
186                     msg = str(_thread.get_native_id())+"\tFLUSH\tRequested"
187                     logging.info(msg)
188                     caching.clean()
189                     c.send(b'200 HTTP OK')
190                 case "DELETE":
191                     try:
192                         caching.delete(reqsplit[2])
193                     except:
194                         c.send(b'404 Not Found')
195                         c.close()
196                         exit()
197             c.send(b'200 HTTP OK')

```



## 2.3 CACHE

Após a implementação do cliente-servidor, desenvolvemos em seguida a implementação do cache. Escolhemos utilizar a base de dados “Last Recently Used”, pois o objetivo dele é remover o dado mais antigo do cache com o intuito de dar mais espaço de armazenamento caso precise, pois também tínhamos que pensar em uma forma de lidar com o tamanho disponível de cache e com esta base de dados pronta facilitaria os tratamentos de erros futuros.

Para realizarmos a implementação fizemos uma verificação, da qual se analisava a url requerida, se esta requisição fosse válida e se estivesse em cache ela seria retornada, mas caso não estivesse o dado é setado dentro do cache com um tempo definido para ser expirado.

```
12 class LRUCache(object):
13     def __init__(self, tam):
14         self.tam = tam
15         self.cache = {}
16         self.lru = {}
17         self.expires = {}
18         self.tm = 0
```

```

20     def get(self, key):
21         if key in self.cache:
22             self.lru[key] = self.tm
23             self.tm = self.tm + 1
24             return self.cache[key]
25         else:
26             return -1
27
28     def set(self, key, value):
29         sizebytes = 0
30         for i in self.cache:
31             sizebytes += sys.getsizeof(self.cache[i])
32         while (sizebytes + sys.getsizeof(value)) > self.tam:
33             if sys.getsizeof(value) > self.tam:
34                 msg = str(_thread.get_native_id()) + "\tEVICT\t"+key+"\tCACHE FULL"
35                 logging.info(msg)
36             else:
37                 msg = str(_thread.get_native_id()) + "\tEVICT\t"+key+"\tCACHE FULL"
38                 logging.info(msg)
39                 old_key = min(self.lru.keys(), key=lambda k: self.lru[k])
40                 msg = str(_thread.get_native_id()) + "\tEVICT\t"+old_key+"\tEXPIRED"
41                 logging.info(msg)
42                 self.cache.pop(old_key)
43                 self.lru.pop(old_key)
44                 self.expires.pop(old_key)
45             sizebytes = 0
46             for i in self.cache:
47                 sizebytes += sys.getsizeof(self.cache[i])
48         self.cache[key] = value
49         self.lru[key] = self.tm
50         tempo = datetime.datetime.now()
51         tempo += datetime.timedelta(minutes=1)
52         self.expires[key] = tempo
53         self.tm = self.tm + 1

```

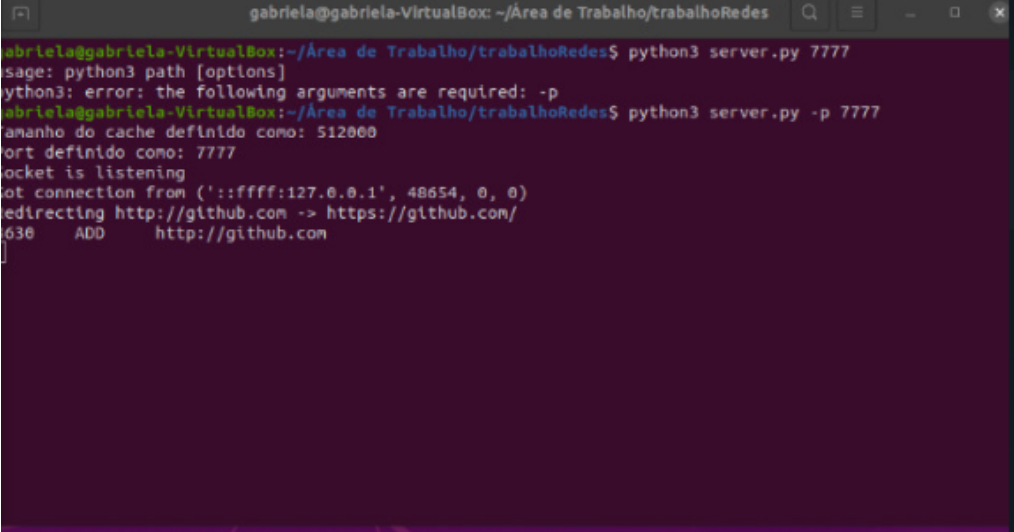
### 3. DECISÕES DE IMPLEMENTAÇÕES QUE PORVENTURA ESTEJAM OMISSOS NA ESPECIFICAÇÃO

No trabalho havia diversas especificações, das quais podiam ser implementadas e dado a possibilidade resolvemos implementar as estatísticas que consiste em mostrar *número total de pedidos*, *número total de hits*, *número total de fails* e *tamanho médio das páginas em cache*.

```
case "INFO":
    match reqsplit[2]: # TODO
        case "0": # salva tamanho atual e lista de objetos do cache
            caching.dump()
            c.send(b'200 HTTP OK')
        case "1": # salva os não-expirados (igual o 0)
            caching.dumpnotex()
            c.send(b'200 HTTP OK')
        case "2": # mostra as estatísticas
            sizebytes = 0
            for i in caching.cache:
                sizebytes += sys.getsizeof(caching.cache[i])
            msg = str(_thread.get_native_id()) + "\tNúmero Total de Pedidos:\t"+str(numpedidos)
            logging.info(msg)
            msg = str(_thread.get_native_id()) + "\tNúmero Total de Hits:\t"+str(numhits)
            logging.info(msg)
            msg = str(_thread.get_native_id()) + "\tNúmero Total de Fails:\t"+str(numfails)
            logging.info(msg)
            try:
                msg = str(_thread.get_native_id())+"\tTamanho Médio das Páginas em Cache:\t"+str((sizebytes/numfails)/1024)
                logging.info(msg)
            except:
                msg = str(_thread.get_native_id())+"\t0 Páginas em Cache"
                logging.info(msg)
            c.send(b'200 HTTP OK')
        case other:
            c.send(b'Error 501 Not Implemented!')
```

## 4. COMO FOI TRATADA A RETRANSMISSÃO DE MENSAGEM

A retransmissão de mensagem só é feita após a conexão do cliente a partir da porta vinculada ao servidor, do qual a porta nós escolhemos. Após o servidor entender que há uma conexão, uma mensagem retorna informando que IP e porta foram conectados.

A terminal window titled 'gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes' shows the execution of a Python script. The user runs 'python3 server.py 7777', which results in an error: 'python3: error: the following arguments are required: -p'. The user then runs 'python3 server.py -p 7777', which successfully starts the server. The output shows the port defined as 7777, the socket is listening, and a connection from '127.0.0.1' is received. The server then responds with '200 HTTP OK' and 'redirecting http://github.com -> https://github.com/636 ADD http://github.com'.

```
gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes$ python3 server.py 7777
python3: error: the following arguments are required: -p
gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes$ python3 server.py -p 7777
port definido como: 7777
socket is listening
got connection from ('::ffff:127.0.0.1', 48654, 0, 0)
redirecting http://github.com -> https://github.com/
636 ADD http://github.com
```

Quando for a requisição de um cliente que seja realizada com sucesso, aparecerá uma mensagem: “200 HTTP OK” e esta mensagem aparecerá para todas as requisições que forem um sucesso, seja ela ADMIN INFO ou DELETE.

```
gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes
meta name="octolytics-url" content="https://collector.github.com/github/collect" />

<meta name="user-login" content="">

<meta name="viewport" content="width=device-width">

<meta name="description" content="GitHub is where over 83 million developers shape the future
of software, together. Contribute to the open source community, manage your G
gabriela@gabriela-VirtualBox:~/Área de Trabalho/trabalhoRedes$ python3 client.py 7777
request: ADMIN INFO 0
00 HTTP OK
```

## 5. TESTES, MOSTRANDO QUE O PROGRAMA ESTÁ FUNCIONANDO DE ACORDO COM AS ESPECIFICAÇÕES SEGUIDOS DA ANÁLISE.

Para a execução do cliente-servidor devemos utilizar linhas de comando no terminal, tais quais são:

### LINHA DE COMANDO - CLIENTE

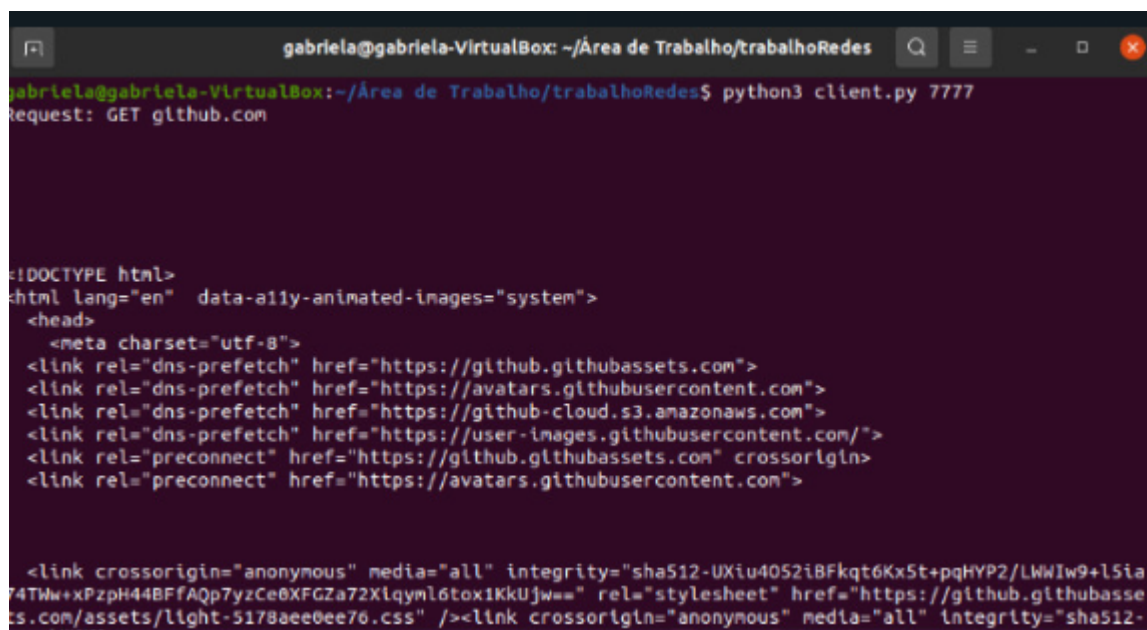
```
gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes$ python3 client.py 7777
```

### LINHA DE COMANDO - SERVIDOR

```
gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes$ python3 server.py -p 7777
```

Nosso trabalho foi desenvolvido para funcionar da seguinte forma:

- 1- Inicializamos o servidor.
- 2- Inicializamos o cliente.
- 3- utilizamos o comando GET na parte do cliente e em seguida colocamos a URL desejada (ex: GET github.com).



```
gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes$ python3 client.py 7777
Request: GET github.com

<!DOCTYPE html>
<html lang="en" data-a11y-animated-images="system">
  <head>
    <meta charset="utf-8">
    <link rel="dns-prefetch" href="https://github.githubassets.com">
    <link rel="dns-prefetch" href="https://avatars.githubusercontent.com">
    <link rel="dns-prefetch" href="https://github-cloud.s3.amazonaws.com">
    <link rel="dns-prefetch" href="https://user-images.githubusercontent.com/">
    <link rel="preconnect" href="https://github.githubassets.com" crossorigin>
    <link rel="preconnect" href="https://avatars.githubusercontent.com">

    <link crossorigin="anonymous" media="all" integrity="sha512-UXiu4052i8Fkqt6Kx5t+pqHYP2/LWWIw9+l5ia
74TWw+XPzPH44BFfAQp7yzCe0XFGZa72Xlqym16tox1KKUjw==" rel="stylesheet" href="https://github.githubasse
ts.com/assets/light-5178aee0ee76.css" /><link crossorigin="anonymous" media="all" integrity="sha512-
LWwIw9+l5ia74TWw+XPzPH44BFfAQp7yzCe0XFGZa72Xlqym16tox1KKUjw==" rel="stylesheet" href="https://github.githubasse
```

```
gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes
gabriela@gabriela-VirtualBox:~/Área de Trabalho/trabalhoRedes$ python3 server.py 7777
usage: python3 path [options]
python3: error: the following arguments are required: -p
gabriela@gabriela-VirtualBox:~/Área de Trabalho/trabalhoRedes$ python3 server.py -p 7777
Tamanho do cache definido como: 512000
Port definido como: 7777
Socket is listening
Got connection from ('::ffff:127.0.0.1', 48654, 0, 0)
Redirecting http://github.com -> https://github.com/
3636  ADD      http://github.com
```

Há duas coisas que podem acontecer quando o proxy atender a requisição:

1- O tamanho do cache inicial ser suficiente (512 KB) e assim concluir a requisição e realizar uma cópia em cache para caso seja feita a mesma requisição ela aconteça mais rapidamente.

2- O tamanho inicial do cache não será o suficiente e consequentemente não irá conseguir realizar uma cópia em cache e o intuito da função que seria economia de tempo ir em vão.

Mas, para resolver isto basta utilizar a função CHANGE que permite que o usuário modifique o tamanho base do cache e consequentemente conseguir guardar as informações necessárias e atingindo o objetivo que é economizar tempo, com esta opção funcionando de maneira adequada pode acabar nos poupando em média 0,10 ms de tempo de requisição.

```
gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes

<meta name="user-login" content="">

<meta name="viewport" content="width=device-width">

<meta name="description" content="GitHub is where over 83 million developers shape the future
of software, together. Contribute to the open source community, manage your G
gabriela@gabriela-VirtualBox:~/Área de Trabalho/trabalhoRedes$ python3 client.py 7777
Request: ADMIN INFO 0
200 HTTP OK

gabriela@gabriela-VirtualBox:~/Área de Trabalho/trabalhoRedes$ python3 client.py 7777
Request: ADMIN CHANGE 80000
```

```
gabriela@gabriela-VirtualBox: ~/Área de Trabalho/trabalhoRedes

gabriela@gabriela-VirtualBox:~/Área de Trabalho/trabalhoRedes$ python3 server.py -p 7777
tamanho do cache definido como: 512000
port definido como: 7777
socket is listening
got connection from ('::ffff:127.0.0.1', 48660, 0, 0)
redirecting http://github.com -> https://github.com/
068 ADD http://github.com
got connection from ('::ffff:127.0.0.1', 48662, 0, 0)
redirecting http://github.com -> https://github.com/
095 HIT http://github.com
got connection from ('::ffff:127.0.0.1', 48664, 0, 0)
redirecting http://github.com -> https://github.com/
162 HIT http://github.com
got connection from ('::ffff:127.0.0.1', 48666, 0, 0)
226 DUMP Dump Start
226 DUMP Size 293.2451171875
226 DUMP fileid1 293.2451171875 2022-08-18 17:11:15.938502 http://github.com
226 DUMP Dump End
got connection from ('::ffff:127.0.0.1', 48668, 0, 0)
287 CHSIZE old: 500.0 new: 80000
```



A seguir, iremos demonstrar as funções REMOVE, FLUSH, INFO 0, INFO 1, INFO 2 a partir do registro de log em funcionamento:

```
1 Got connection from (::ffff:127.0.0.1', 48660, 0, 0)
2 Redirecting http://github.com -> https://github.com/
3 4068 ADD http://github.com
4 Got connection from (::ffff:127.0.0.1', 48662, 0, 0)
5 Redirecting http://github.com -> https://github.com/
6 4095 HIT http://github.com
7 Got connection from (::ffff:127.0.0.1', 48664, 0, 0)
8 Redirecting http://github.com -> https://github.com/
9 4162 HIT http://github.com
10 Got connection from (::ffff:127.0.0.1', 48666, 0, 0)
11 4226 DUMP Dump Start
12 4226 DUMP Size 293.2451171875
13 4226 DUMP fileid1 293.2451171875 2022-08-18 17:11:15.938502 http://github.com
14 4226 DUMP Dump End
15 Got connection from (::ffff:127.0.0.1', 48668, 0, 0)
16 4287 CHSIZE old: 500.0 new: 80000
17 Got connection from (::ffff:127.0.0.1', 48670, 0, 0)
18 Got connection from (::ffff:127.0.0.1', 48672, 0, 0)
19 Got connection from (::ffff:127.0.0.1', 48674, 0, 0)
20 Redirecting http://github.com -> https://github.com/
21 4476 HIT http://github.com
22 Got connection from (::ffff:127.0.0.1', 48676, 0, 0)
23 Got connection from (::ffff:127.0.0.1', 48678, 0, 0)
24 Got connection from (::ffff:127.0.0.1', 48680, 0, 0)
25 4642 DUMP Dump Start
26 4642 DUMP Size 0.0
27 4642 DUMP Dump End
28 Got connection from (::ffff:127.0.0.1', 48682, 0, 0)
29 4708 FLUSH Requested
30 Got connection from (::ffff:127.0.0.1', 48684, 0, 0)
31 4762 Número Total de Pedidos: 13
32 4762 Número Total de Hits: 3
33 4762 Número Total de Fails: 1
34 4762 Tamanho Médio das Páginas em Cache: 0.0
```

## **6.CONCLUSÕES E REFERÊNCIAS BIBLIOGRÁFICAS**

Com o passar do tempo que ficamos para realizar o desenvolvimento deste trabalho de forma usual, nos deparamos com várias conclusões, pois a cada etapa concluída do projeto conseguimos visualizar de maneira mais clara onde poderíamos aplicá-lo na vida fora do meio acadêmico.

Conseguimos compreender completamente as questões de segurança que envolvem um proxy e nas vantagens que isto nos traz, pois daria para ser implementado um sistema do qual de onde sequer a pessoa fosse acessá-lo ainda conseguimos o endereço de IP, fazendo com que assim seja mais eficaz o sistema de detecção de violação de ética dentro de uma empresa ou qualquer outro tipo de estabelecimento. Com este projeto podemos também definir os níveis de acesso do usuário de maneira extremamente eficaz, pois ficaria tudo registrado no log.

O processo de desenvolvimento foi difícil, porém conseguimos adquirir e aprimorar nosso conhecimento em Sockets e na linguagem de programação Python de maneira do qual nos sentimos preparados para implementar um sistema desse em qualquer lugar que nos fosse pedido, conseguimos ver uma utilidade fora da curva que com certeza apenas nos acrescentou seja dentro do meio acadêmico quanto no mercado de trabalho. Pode ter sido difícil, mas não podemos ser injustos também foi um projeto extremamente divertido de ser desenvolvido pois ver as funções funcionando e entendendo de ponta a ponta como um proxy e um cache funciona foi magnífico.

No final de tudo, ficamos completamente agradecidos por termos conseguido concluir um trabalho deste nível pois no início não tínhamos noção de sua complexidade.

## REFERÊNCIAS BIBLIOGRÁFICAS

<https://www.keycdn.com/support/if-modified-since-http-header#:~:text=The%20If-Modified-Since%20HTTP,last%20time%20it%20was%20accessed>

<https://www.geeksforgeeks.org/clear-lru-cache-in-python/>

[https://www.youtube.com/watch?v=FExXpnu9u8&ab\\_channel=SoumilShah](https://www.youtube.com/watch?v=FExXpnu9u8&ab_channel=SoumilShah)

<https://www.geeksforgeeks.org/lru-cache-in-python-using-ordereddict/>

<https://www.analyticsvidhya.com/blog/2021/08/caching-in-python-the-lru-algorithm/>

<https://refactoring.guru/pt-br/design-patterns/proxy/python/example>

<https://realpython.com/command-line-interfaces-python-argparse/>

<https://stackoverflow.com/questions/42908453/python-web-proxy-failing-to-load-https-sites>

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods/CONNECT>