Projeto Final - Matheus Martins

Relatório do projeto

Autor: Matheus Martins da Silva
Localização do projeto no GitHub: https://github.com/MatheusMartins1/AT_MATHEUS_MARTINS

Projeto referente a disciplina de projeto de bloco em arquitetura de computadores, sistemas operacionais e redes da graduação em Sistemas de Informação apresentado ao Instituto INFNET como requisito para a obtenção de grau na Atividade proposta.

Descrição do projeto

O entregável do seu projeto de bloco será: Um software cliente-servidor em Python que explore conceitos de arquitetura de redes, arquitetura de computadores e/ou de sistemas operacionais, acompanhado de relatório explicativo.

Entregável

Um aplicativo simples de apresentação gráfica do monitoramento e análise do computador. Ele deverá ser implementado em Python usando módulos como 'psutil', 'cpuinfo', 'os' e 'subprocess' (para capturar dados do sistema computacional, informações dos diretórios e processos) e 'pygame' (para exibir graficamente os dados).

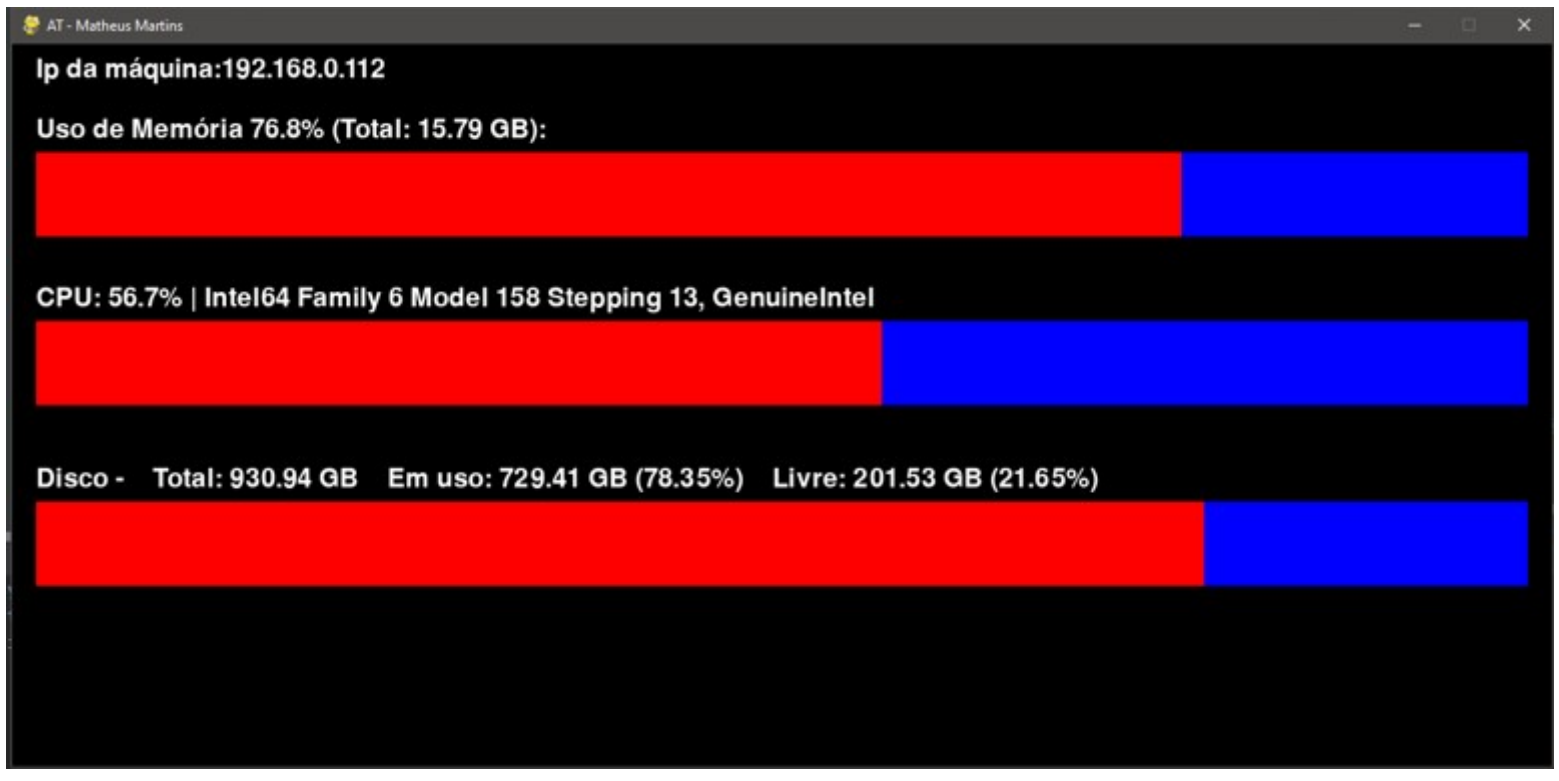
Mudanças acordadas

Apesar do entregável oficial do projeto solicitar a exibição gráfica através do pygame, foi acordado com o professor de bloco a exibição gráfica do projeto utilizando Django.
Até o TP5 o projeto foi construído progressivamente utilizando o pygame como estrutura gráfica padrão, a partir do TP6 o projeto foi drasticamente alterado para implantar a aplicação em Django

TESTE DE PERFORMANCE - TP2

Escreva um programa em Python que exiba graficamente através do uso do módulo Pygame:

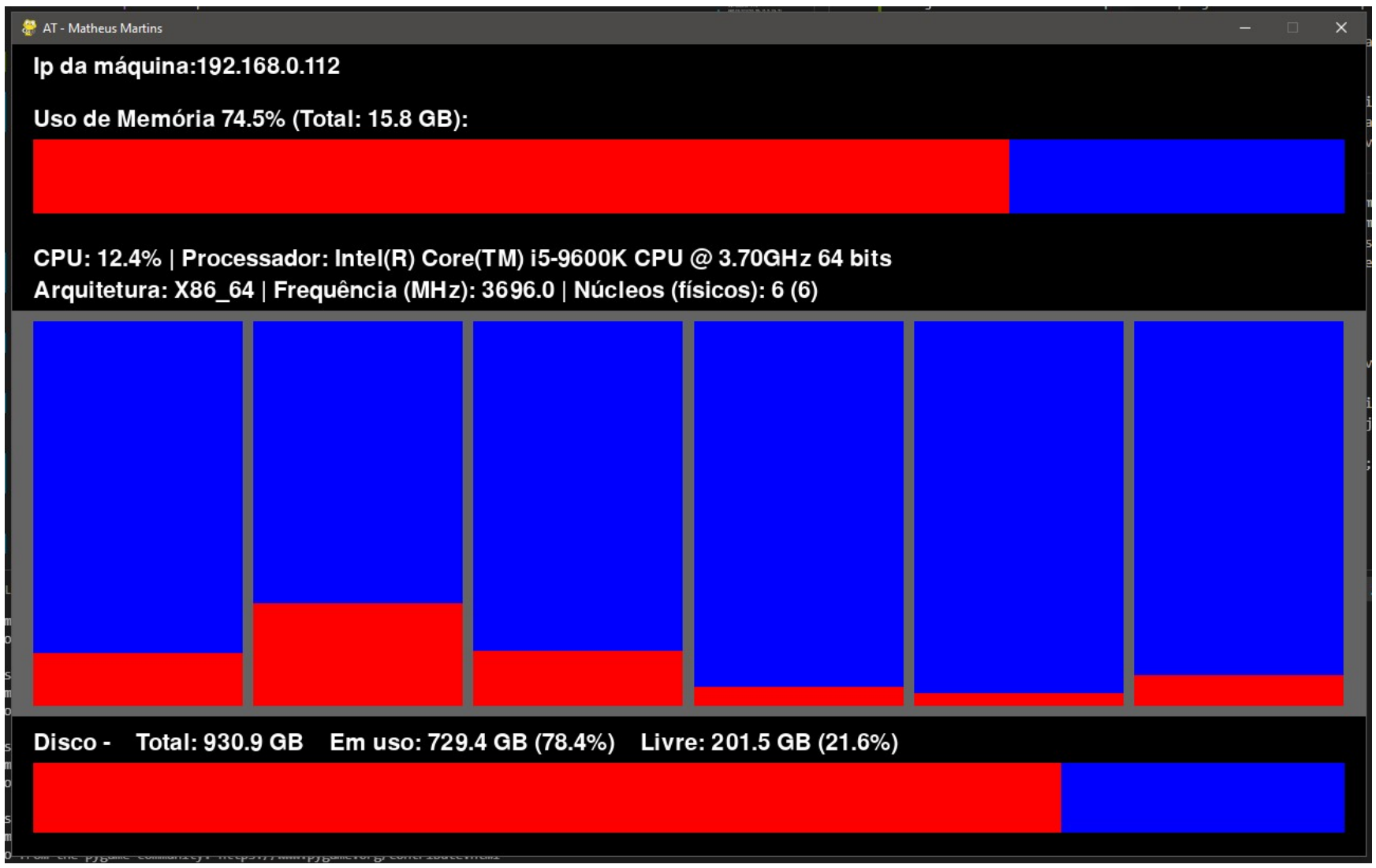
1. Uma barra de indicação da porcentagem do uso de memória;
2. Uma barra de indicação da porcentagem do uso de CPU, junto com a informação detalhada da plataforma de processamento;
3. Uma barra de indicação da porcentagem do uso de disco;
4. Um texto com a informação do IP da máquina.



TESTE DE PERFORMANCE - TP3

Adicione ao seu programa feito no TP2 informações mais detalhadas de uso de CPU. Você pode adicionar as informações da maneira que achar mais interessante. No entanto, algumas são obrigatórias: Mantenha as barras de uso de memória e disco (e suas superfícies). Elas serão alteradas, da mesma forma que a CPU, em TPs posteriores.

1. Alterar o programa feito no TP2 de modo a possuir 5 visualizações diferentes:
    1.1. Uma para colocar todas as informações associadas ao processador
    1.2. Uma para colocar todas as informações associadas à memória
    1.3. Uma para colocar todas as informações associadas ao Disco
    1.4. Uma para colocar todas as informações associadas ao IP
    1.5. Todas as anteriores devem ser alteradas caso o usuário clique nas setas esquerda ou direita do teclado. Seguindo sempre uma ordem predefinida, como em um carrossel.
    1.6. Uma última que teria um resumo de todas elas. O qual seria acessado quando o usuário clica na tecla “Barra” do teclado.
2. Alterar a barra de CPU do TP2 para ter barras de CPU associadas a cada núcleo (core);
3. Adicionar informação de nome/modelo da CPU (brand);
4. Adicionar informação do tipo da arquitetura (arch);
5. Adicionar informação da palavra do processador (bits);
6. Adicionar informação sobre a frequência total e frequência de uso da CPU;
7. Adicionar informação do número total de núcleos (núcleo físico) e threads (núcleo lógico).



TESTE DE PERFORMANCE - TP4

Este TP4 corresponde à continuação do TP3. Agora, você irá introduzir no seu programa informações sobre arquivos e diretórios especificados e sobre processos em execução no computador. A partir de então, você terá mais liberdade para criar a forma de visualização das informações da maneira que desejar e de acordo com seu orientador. Certifique-se apenas que está realizando o básico do que este TP requisita.

1. Criar uma ou mais funções que retornem ou apresentem informações sobre diretórios e arquivos. Tais informações podem ser qualquer uma que você achar relevante disponível no módulo 'os' e 'psutil' de Python, como nome, tamanho, localização, data de criação, data de modificação, tipo, etc.
2. Usar a função em seu programa para mostrar o resultado. O resultado pode ser em texto formatado impresso na tela ou gráfico, usando o módulo 'pygame'. Note que o uso do 'pygame' é opcional.

```
"C:\Program Files\Python38\python.exe" D:/Users/Matheus/Documents/workspacePython/AT_PB_MATHEUS_MARTINS/main.py
pygame 2.0.1 (SDL 2.0.14, Python 3.8.3)
Hello from the pygame community. https://www.pygame.org/contribute.html

Tamanho    Data de Modificação        Data de Criação            Nome       tipo
121.71 KB   2021-06-28 22:28:20   2021-06-13 17:17:31   .git   pasta
34.00 B   2021-06-28 21:54:43   2021-06-13 16:55:57   .gitignore   pasta
7.28 KB   2021-06-28 22:28:20   2021-06-28 22:26:55   .idea   pasta
12.81 KB   2021-06-28 20:49:08   2021-05-31 09:00:36   aulas_PB   py
2.68 KB   2021-06-28 20:49:08   2021-06-13 16:11:10   cpu   py
153.64 KB   2021-06-28 22:28:20   2021-06-13 17:10:11   files   pasta
7.57 KB   2021-06-28 22:28:19   2021-06-28 22:28:18   main   py
9.47 KB   2021-06-26 20:19:03   2021-06-13 17:16:54   README   md
13.77 MB   2021-06-28 22:28:20   2021-06-08 08:49:06   venv   pasta
```

1. Criar uma ou mais funções que retornem ou apresentem informações sobre processos do sistema. As informações podem ser: PID, nome do executável, consumo de processamento, consumo de memória, entre outras disponíveis no módulo 'psutil' de Python.
2. Usar a função em seu programa para mostrar o resultado. O resultado pode ser em texto formatado impresso na tela ou gráfico, usando o 'pygame'. Note que o uso do 'pygame' é opcional.

```
Neste computador existem 291 processos em execução;
Destes, 46 processos (15.8%) são do Navegador Google Chrome;
O Chrome está utilizando 27.42% (4.33 GB) de memória ram em um total de 15.79 GB.

Exemplo de processo do WhatsApp em execução:
{'dt_inicio': '2021-06-28 07:13:30',
 'memoria': 0.0621803229365159,
 'nome': 'WhatsApp.exe',
 'pid': 13912,
 'status': 'running',
 'tempo_exec': '12:31:52',
 'tempo_system': '22:45:22',
 'tempo_user': '22:45:22',
 'threads': 8}
```

TESTE DE PERFORMANCE - TP5

1. Utilizar o módulo 'sched' para chamar as funções criadas no TP4 que retornam as informações sobre diretórios e arquivos.
   Nesta etapa foi adicionado um botão chamado "Clique aqui" que utiliza o módulo 'sched' para executar as funções de processos em execução no computador.

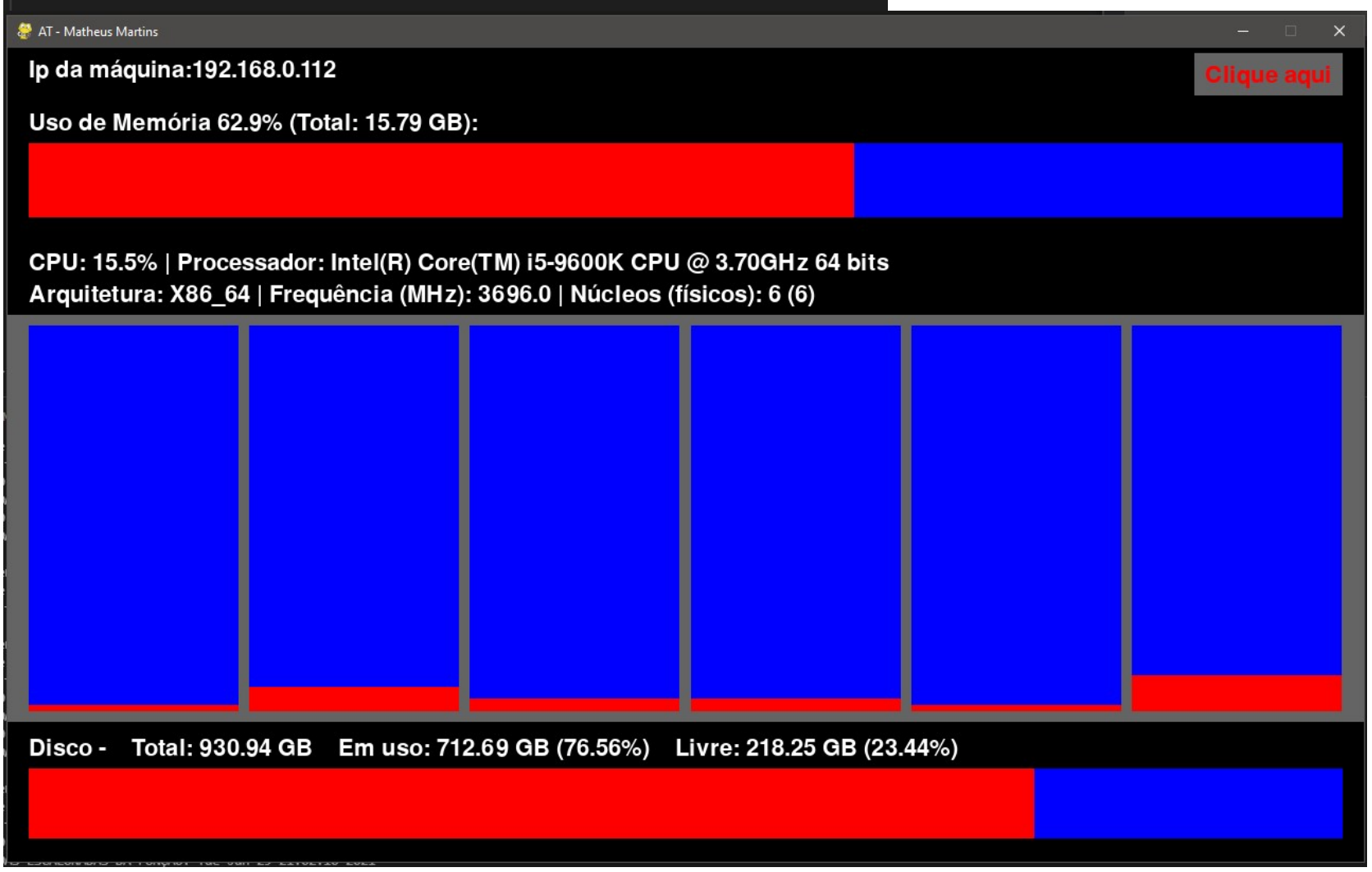
```python
for event in pygame.event.get():

    if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
        point = pygame.mouse.get_pos()

        # clique dentro do quadrado
        if quadrado.collidepoint(point):
            tela.fill(PRETO)
            scheduler = sched.scheduler(time.time, time.sleep)

            inicio = datetime.now()
            print('INICIO:', time.ctime())
            scheduler.enter(5, 2, formata_diretorio)
            scheduler.enter(1, 1, retorna_processo)

            scheduler.run()
            print('Fim:', time.ctime())
            print('Duração:', datetime.now() - inicio)
            posi = 0
```



2. Realizar um escalonamento das chamadas das funções com o módulo 'sched' e medir o tempo total utilizado por cada chamada com o módulo 'time'. Você pode escolher com quais funções do seu projeto realizar o escalonamento, deixando indicado no relatório.
   Ao clicar no botão, a tela do pygame é alterada para mostrar informações sobre os processos e acontece o escalonamento:

3. Dentro do escalonamento realizado na questão anterior, realizar uma comparação dos tempos obtidos com a quantidade total de clocks utilizados pela CPU para a realização dessas mesmas chamadas.
4. Indique a diferença de tempo real e tempo do clock do computador.
5. Indique o que acontece com essa diferença quando insere um tempo de espera, como por exemplo utilizando o 'time.sleep' dentro de alguma chamada.

```
D:\Users\Matheus\Documents\workspacePython\AT_PB_MATHEUS_MARTINS>"C:/Program Files/Python38/python.exe" d:/Users/Matheus/Documents/workspacePython/AT_PB_MATHEUS_MARTINS/main.py
pygame 2.0.1 (SDL 2.0.14, Python 3.8.3)
Hello from the pygame community. https://www.pygame.org/contribute.html
INICIO: Tue Jun 29 21:46:25 2021
Fim: Tue Jun 29 21:46:30 2021
Duração: 0:00:05.085711
```

Implantação do Django

A apartir deste momento o Pygame foi descontinuado e o Django foi implementado.

Até o momento todas as funções do projeto constavam dentro da main com o pygame. Considerando as boas práticas de programação, as funções da aplicação foram separadas em diferentes arquivos a serem importados como módulos durante a execução necessária. Além disso foram criados dois arquivos, "CONSTANTES.py" e "funcoes.py" que possuem variáveis e funções centralizadas que são usadas nos demais módulos.

Afim de evitar retrabalhos e deixar o desenvolvimento mais fluído, a ordem de construção do roteiro padrão foi alterado. Antes do desenvolvimento do TP6 foi desenvolvido os sockets do TP8 e integrado ao servidor do Django. Todas as funções do projeto também foram redesenhadas para serem usadas nas integrações HTTP do Django.

Na estrutura do Django, foram criados as seguintes pastas:

⊙ AT_MATHEUS_MARTINS

Pacote principal do Django onde estão contidos arquivos de configurações gerais do projeto, como: settings, urls (que define as rotas web do projeto), asgi e wsgi.

⊙ gerenciador_pc

Pacote interno do projeto, onde estão configurações de views, pasta com arquivos estáticos a serem usados no front-end e o cliente socket usado na aplicação para receber informações do servidor.

neste pacote também contém o arquivo views.py, que resgata as informações do socket cliente e envia em formato JSON p as páginas html carregarem no front-end.

O arquivo views.py também gerencia as rotas e requisições HTTP dentro da aplicação.

⊙ socket_server

Pasta com o servidor socket e com os arquivos contendo as funções de coleta de dados do computador.

Neste pacote contém os modulos com todas as função de coleta de dados. Elas são importadas pelo servidor e gerenciados automaticamente de acordo com as requisições do cliente.

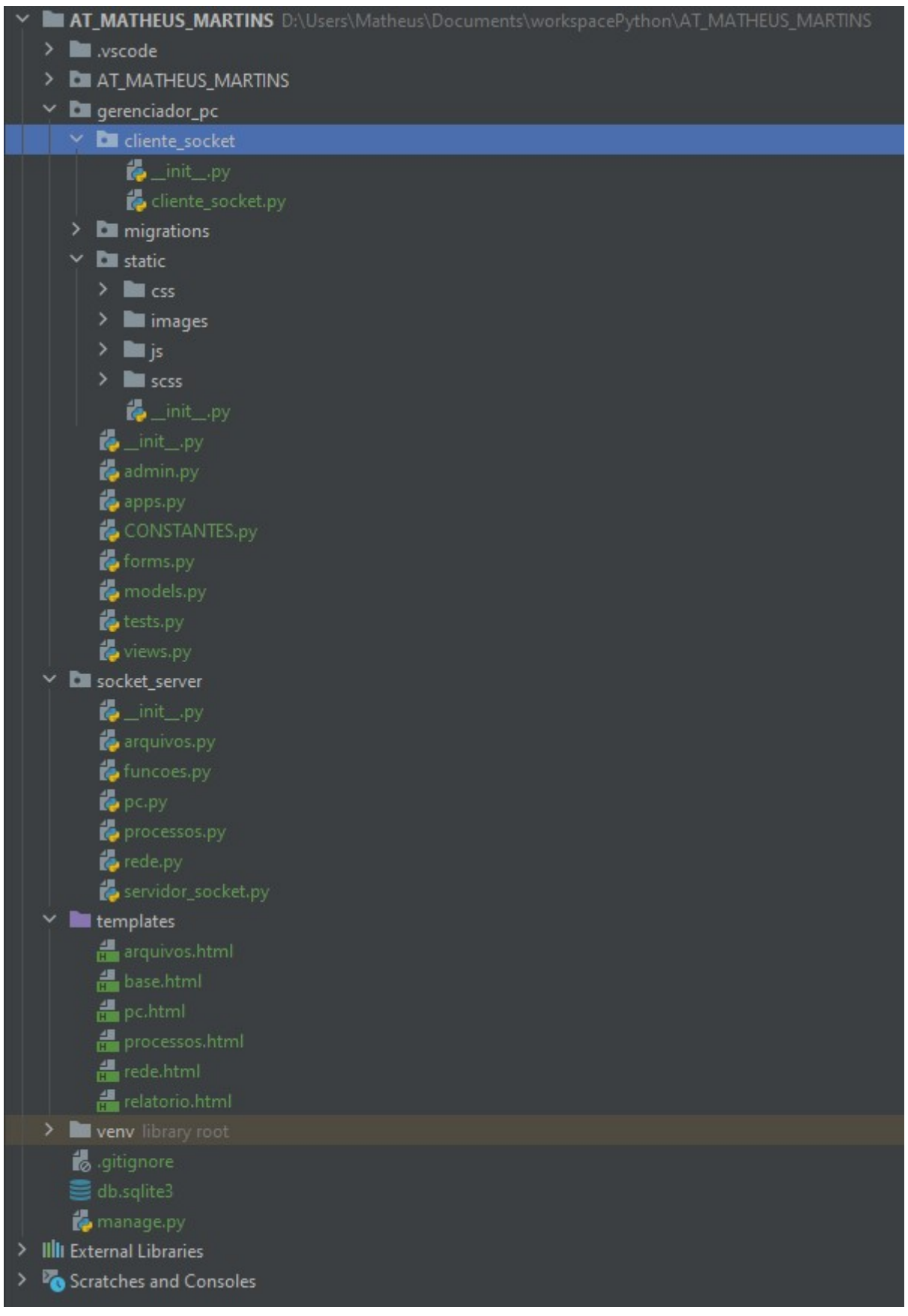
⊙ templates

Páginas html em Django que exibem as informações da aplicação.

⊙ venv

Virtual Environment do python 3.9 contendo todos os módulos externos usados na aplicação

Print da estrutura do projeto:



TESTE DE PERFORMANCE - TP6

Este TP6 corresponde à continuação do TP5. Agora, você irá introduzir no seu programa informações sobre subrede de um IP especificado e sobre as portas desse IP. A partir de então, você terá mais liberdade para criar a forma de visualização das informa da maneira que desejar e de acordo com seu orientador. Certifique-se apenas que está realizando o básico do que este TP requis

1. Criar uma ou mais funções que retornem ou apresentem informações sobre as máquinas pertencentes à sub-rede do IP específico.
2. Usar a função em seu programa para mostrar o resultado. O resultado pode ser em texto formatado impresso na tela ou gráfico, usando o módulo 'pygame'.
3. Criar uma ou mais funções que retornem ou apresentem informações sobre as portas dos diferentes IPs obtidos nessa sub rede.
4. Usar a função em seu programa para mostrar o resultado. O resultado pode ser em texto formatado impresso na tela ou gráfico, usando o 'pygame'.

retorna_codigo_ping()

Define função que realiza um ping para determinado IP. Esta função é executada ao verificar atividade dos hosts em verifica_hosts()

verifica_hosts()

Verifica todos os host dentro da sub_rede entre, faz um loop entre 1 e 255 retorna uma lista com todos os host que tiveram resposta 0 (ativo) utilizando a função retorna_codigo_ping()

scan_host()

Realiza um mapeamento da rede usando nmap, resgata informações de portas abertas e dados dos dispositivos conectados à estas portas. Esta função executa e preenche um dicionário previamente populado pelos hosts ativos identificados na função verifica_hosts().

exec_info_redes()

Função responsável definir a ordem de execução e a das funções definidas anteriormente e popular o dicionário que será retornado para a main da aplicação.

retorna_info_hosts()

Função preventiva de erro, na qual executa a exec_info_redes() se o dicionário de retorno à main esteja vazio. Caso o dicionário esteja preenchido ela o retorna.

Devido ao extenso tempo de carregamento necessário para realizar o mapeamento de rede pelo nmap e a verificação via ping dos endereços ativos, foi necessário realizar uma modificação no servidor socket.
Exemplo do tempo de carregamento para mapeamento completo de uma sub_rede simples:

```
D:\Users\Matheus\Documents\workspacePython\AT_MATHEUS_MARTINS\venv\Scripts\python.exe D:/Users/Matheus/Documents/workspacePython/AT_MATHEUS_MARTINS/socket_server/servidor_socket.py
Servidor de nome 127.0.0.2 esperando conexão na porta 9001
iniciando execução
Mapeando
.............
Mapping ready...
----------
escaneando 192.168.0.1
----------
Protocolo : tcp
escaneando porta 22
escaneando porta 23
escaneando porta 80
escaneando porta 1900
----------
escaneando 192.168.0.3
----------
Protocolo : tcp
escaneando porta 8009
escaneando porta 9080
----------
escaneando 192.168.0.111
----------
Protocolo : tcp
escaneando porta 21
escaneando porta 23
escaneando porta 80
escaneando porta 443
escaneando porta 515
escaneando porta 631
escaneando porta 9100
----------
escaneando 192.168.0.112
----------
Protocolo : tcp
escaneando porta 135
escaneando porta 139
escaneando porta 445
escaneando porta 808
escaneando porta 1051
escaneando porta 1060
escaneando porta 1098
escaneando porta 1688
escaneando porta 2869
escaneando porta 5357
O modulo de redes levou: 0:10:14.924938 para mapear a rede e coletar informações de portas para cada IP
```

Para contornar o tempo de carregamento, o servidor passou a ser assíncrono executando em paralelo a função **exec_info_redes()** e as demais funções do servidor. A prioridade maior é da main do próprio servidor (função exec_server()), a thread que retorna as informações de rede é iniciada e encerrada em paralelo.

```
loop = asyncio.get_event_loop()

loop.create_task(exec_info_redes_async())
loop.run_until_complete(exec_server())

loop.close()
```

TESTE DE PERFORMANCE - TP7

Este TP corresponde à sexta entrega do projeto, no qual o entregável será: Um aplicativo simples de apresentação textual do monitoramento e análise de computadores em rede. Ele deverá ser implementado em Python usando módulos como psutil (para capturar dados do sistema computacional) e sockets (para criar cliente e servidor) e será desenvolvido de forma incremental durante o curso. A apresentação gráfica no lado cliente é opcional e pode ser parcial (parte gráfica e parte texto).

1. Crie uma ou mais funções que retornem ou apresentem as seguintes informações de redes: IP, gateway, máscara de subrede.
2. Crie uma ou mais funções que retornem ou apresentem as seguintes informações de redes: Uso de dados de rede por interface.
3. Crie uma ou mais funções que retornem ou apresentem as seguintes informações de redes: Uso de dados de rede por processos.
4. Use as funções em seu programa para mostrar o resultado.

Neste TP foi inserido no modulo de redes a função retorna_interfaces_redes(), que colheta informações de redes e interfaces utilizando o psutil.

Codigo:

```
106  def retorna_interfaces_redes():
107
108      interfaces = psutil.net_if_addrs()
109      status = psutil.net_if_stats()
110      io_status = psutil.net_io_counters(pernic=True)
111
112      nomes = []
113
114      # Obtém os nomes das interfaces
115      for i in interfaces:
116          nomes.append(str(i))
117          redes_ativas[i] = {}
118
119      for i in nomes:
120          for j in interfaces[i]:
121              redes_ativas[i] = {
122                  "rede": i,
123                  "familia": str(j.family),
124                  "address": j.address,
125                  "netmask": "" if j.netmask == None else str(j.netmask),
126                  "broadcast": "" if j.broadcast == None else str(j.broadcast),
127                  "ptp": "" if j.ptp == None else str(j.ptp),
128                  "ativa": "Sim" if status[i].isup else "Não",
129                  "duplex": str(status[i].duplex).split(".")[1],
130                  "speed": f"{status[i].speed} Mb",
131                  "mtu": f"{status[i].mtu} bytes",
132                  "bytes_sent": formata_tamanho(io_status[i].bytes_sent),
133                  "bytes_recv": formata_tamanho(io_status[i].bytes_recv),
134                  "packets_sent": formata_tamanho(io_status[i].packets_sent),
135                  "packets_recv": formata_tamanho(io_status[i].packets_recv),
136                  "errin": formata_tamanho(io_status[i].errin),
137                  "errout": formata_tamanho(io_status[i].errout),
138                  "dropin": formata_tamanho(io_status[i].dropin),
139                  "dropout": formata_tamanho(io_status[i].dropout)
140
141              }
```

Resultado:

Interfaces:

rede	familia	address	netmask	broadcast	ptp	ativa	duplex	speed	mtu	bytes_sent	bytes_recv	packets_sent	packets_recv	errin	errout	dropin	dropout
Ethernet	AddressFamily.AF_INET6	fe80::2c0c:2d25:68df:8f95				Sim	NIC_DUPLEX_FULL	100 Mb	1500 bytes	388.2 MB	1.93 GB	1.79 MB	1.94 MB	0.00 B	0.00 B	0.00 B	0.00 B
Loopback Pseudo-Interface 1	AddressFamily.AF_INET6	::1				Sim	NIC_DUPLEX_FULL	1073 Mb	1500 bytes	0.00 B	0.00 B	0.00 B	0.00 B	0.00 B	0.00 B	0.00 B	0.00 B

TESTE DE PERFORMANCE - TP8

Você deve organizar todas ou uma parte das implementações realizadas nos TPs anteriores para que, ao invés de obter as informações da própria máquina, elas sejam obtidas na máquina servidora e o cliente apenas as exiba. Tanto a forma de entrada dos dados do usuário de seu programa, como a forma de visualização das informações ficam a seu critério e de acordo com seu orientador. Seu programa cliente pode ter uma interface com o usuário para que ele selecione quais informações ele quer exibir ou pode exibir tudo de uma vez.

1. Implementar ao menos 2 tipos de obtenção de informação de um computador, conforme feito nos TPs anteriores, mas agora no servidor. Você pode implementar todas que foram requisitadas nos TPs anteriores, mas não é necessário no momento. No entanto, será necessário na etapa 09.
   1.1 uso de processamento;
   1.2 uso/capacidade de memória;
   1.3 sobre arquivos e diretórios;
   1.4 sobre processos executando;
   1.5 sobre redes/interfaces de redes.
2. Implemente um programa cliente que requisite tais informações ao programa servidor e exiba os dados usando texto formatado ou de forma visual (com PyGame, por exemplo).
3. Implemente um programa servidor que receba tais requisições do cliente, as obtenha na própria máquina e as envie ao cliente.

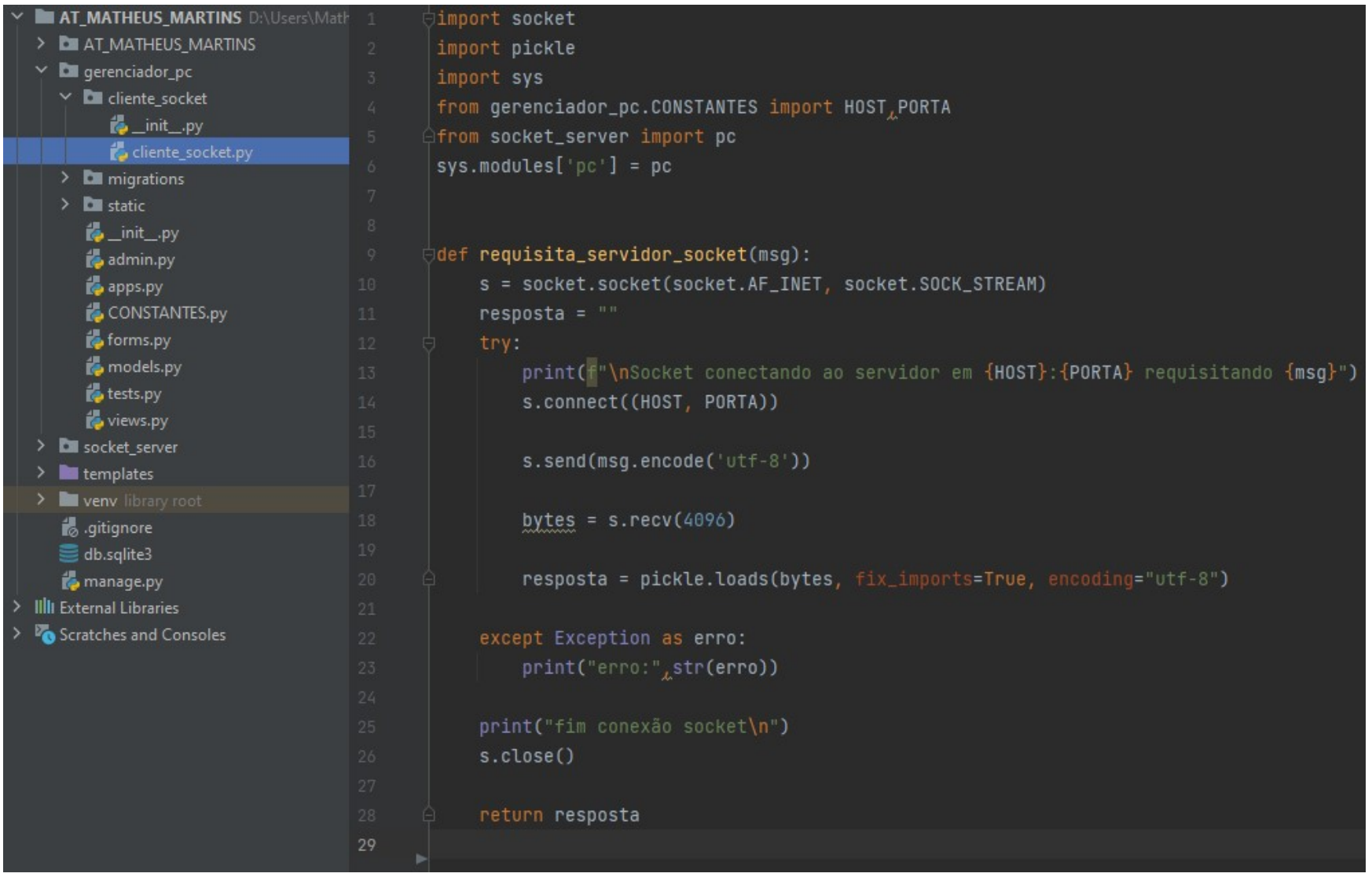
Devido a migração para o Django, esta etapa foi realizada antes do início do TP6, mas a medida que a necessidade foi sugirondo, alterações foram realizadas à ela, como a inclusão de funções assíncronas e solicitações dinâmicas baseada na página carregada no front-end.

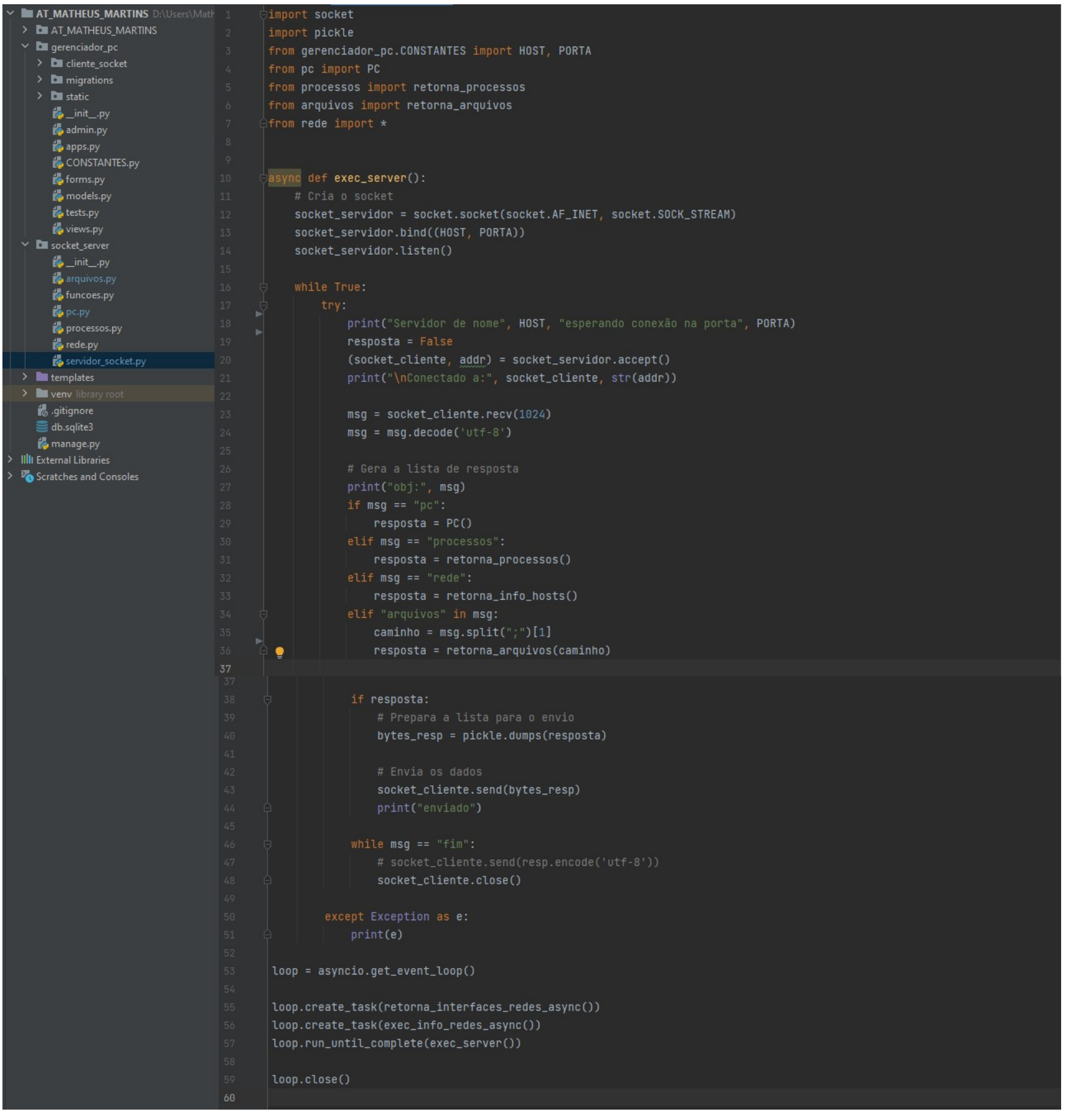
Uma das modificações realizadas no código foi o aumento da quantidade de bytes transmitida via socket para 4096 devido a erros de truncamento de dados em dicionários muito grandes. Mesmo com esse aumento na quantidade de bytes não foi suciciente para tranmitir o dicionário com dados de todos os processos do computador. por isso foi usado uma estratégia de limitar a quantidade de processos respondidos via socket aos 30 primeiros ordenados por consumo de memnória.

Outra adaptação realizada neste módulo foi a inclusão da classe PC de dentro do servidor socket atravpes do comando sys.modules['pc'] para que o pickle conseguisse desempacotar a mensagem transmitida corretamente. Sem este comando não era possível interpretar a classe PC no cliente.

Foi criada uma função que padronizasse a execução do cliente socket, assim o arquivo views.py que gerencia as requisições chama a função do cliente socket passando qual módulo deseja resgatar. Só então o cliente começa a solicitação de transferência, após finalizado a função encerra a conexão e retorna para o views.py a resposta do servidor.

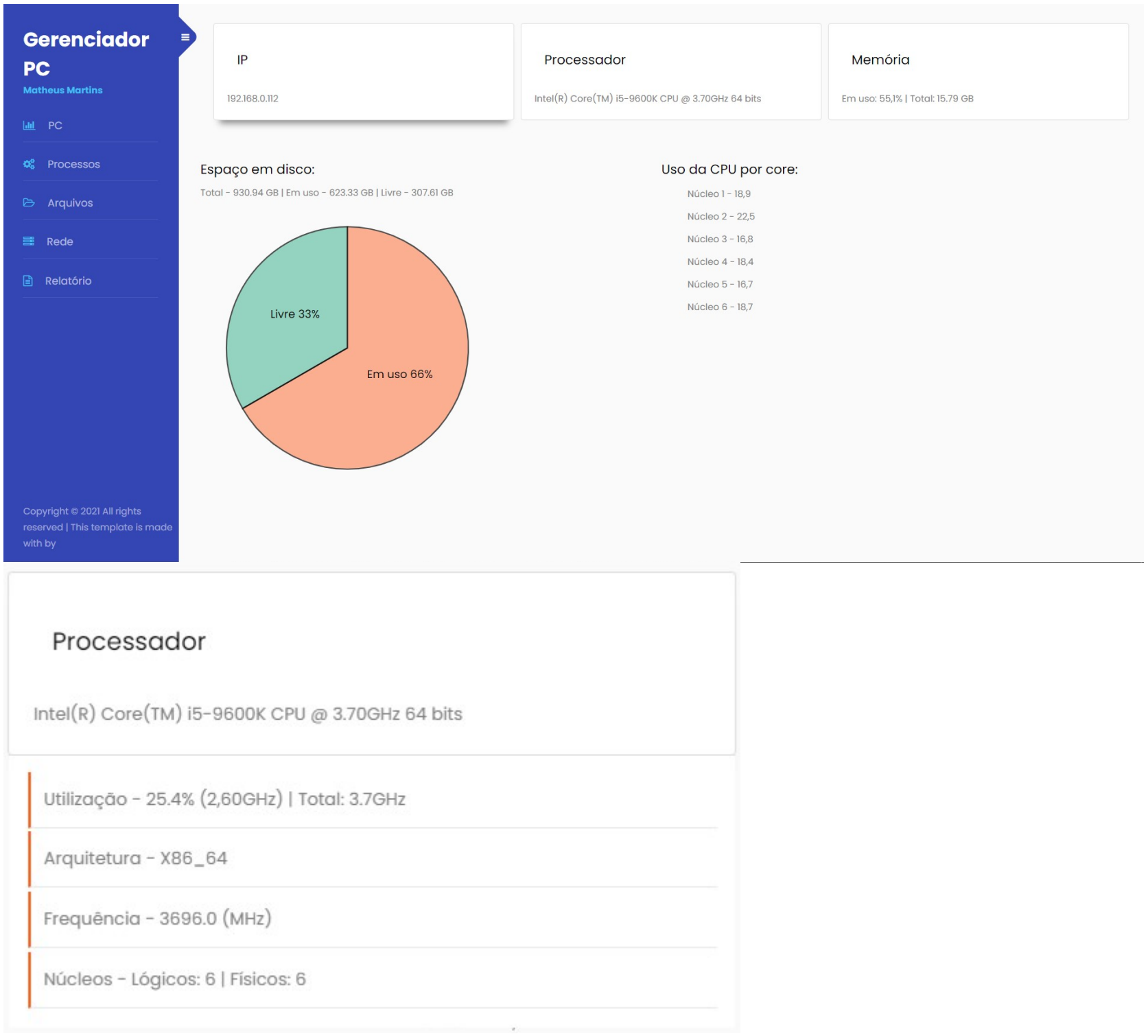
Código do cliente socket que é executado na main do Django:





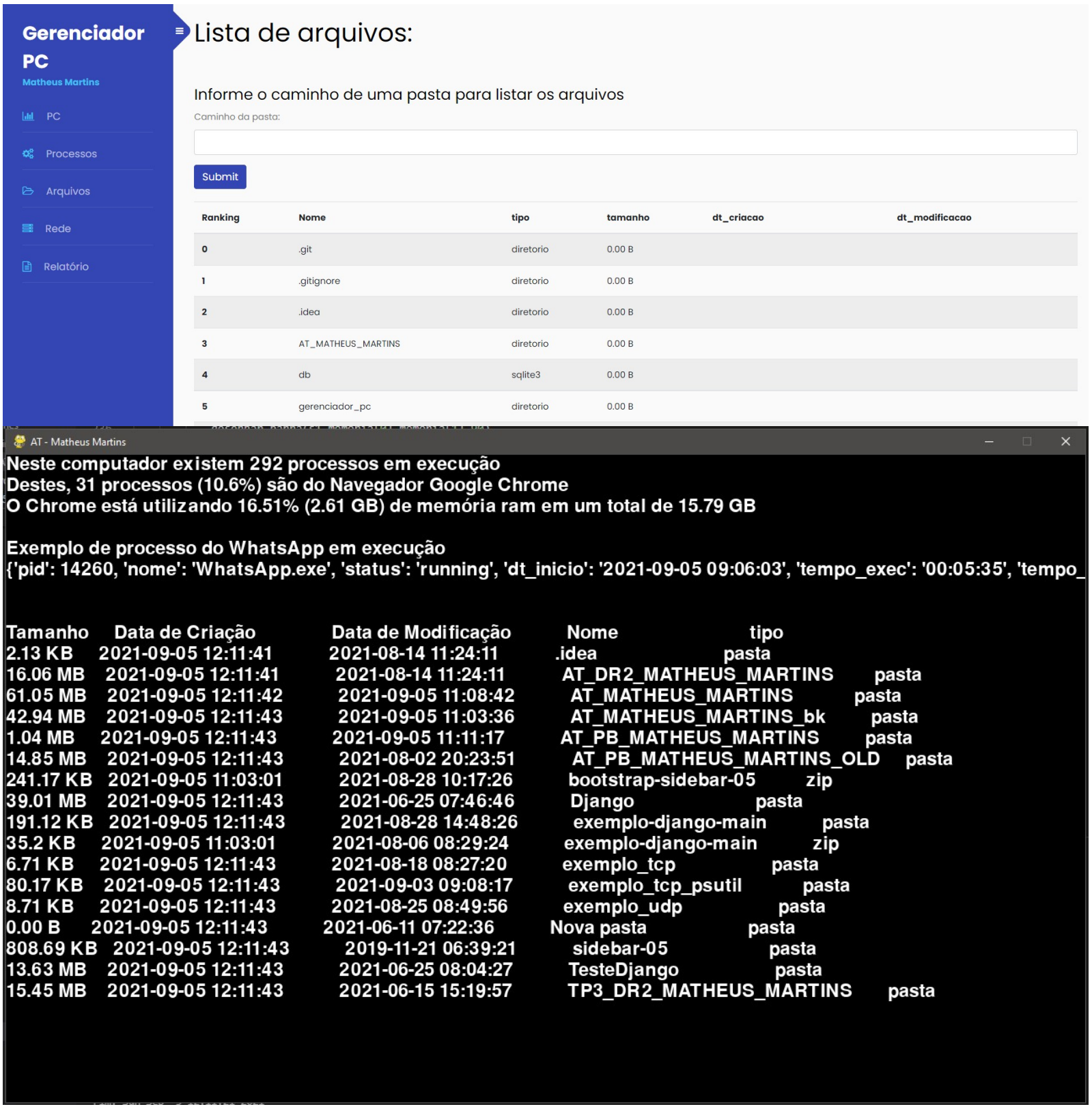
TESTE DE PERFORMANCE - TP9

1. Informações do Computador

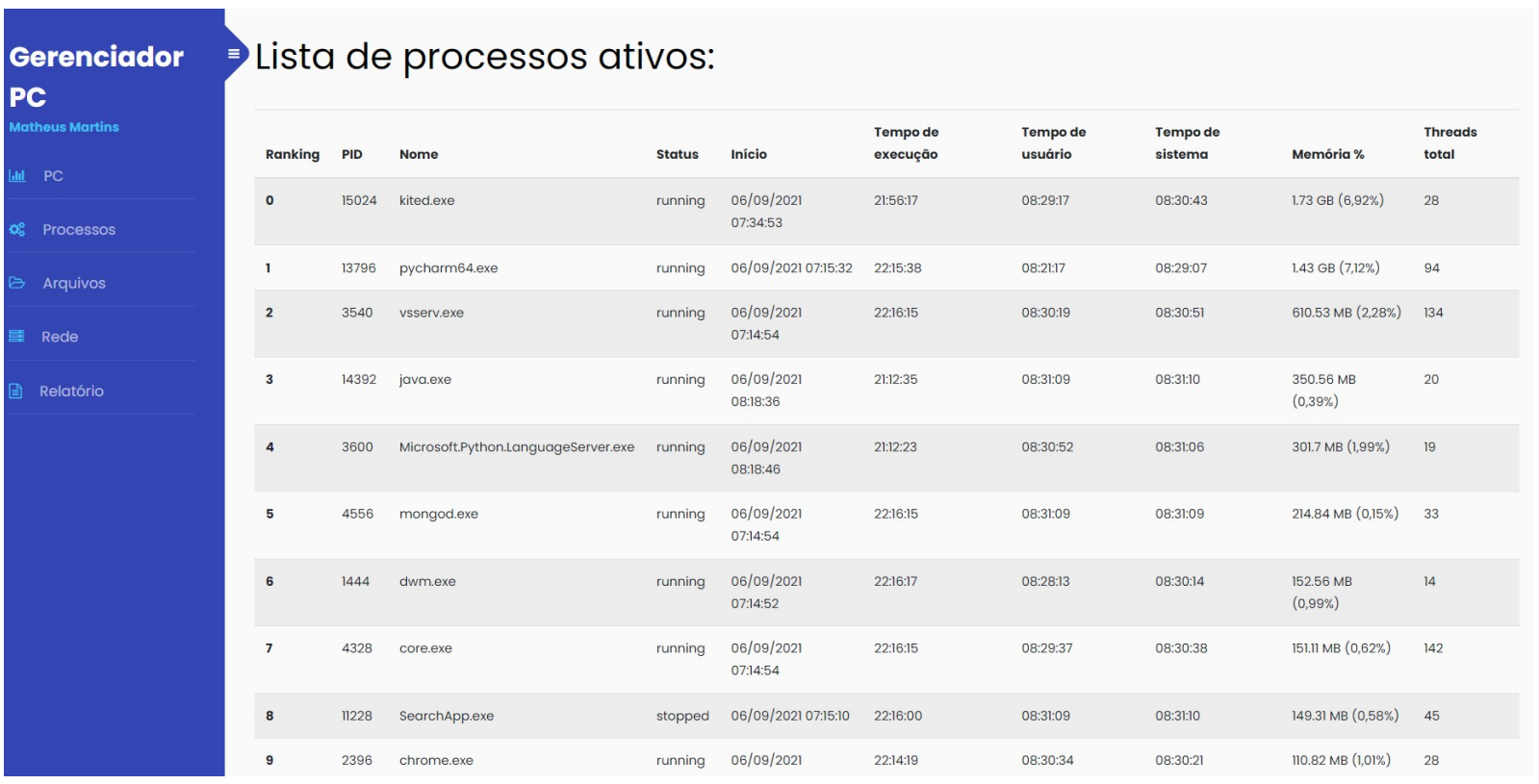


2. Informações necessárias do TP 4:

    Capturas das informações dos diretórios, como nome, tamanho, localização, data de criação, data de modificação, tipo, etc.



1.2. Capturas das informações dos processos do sistema, como PID, nome do executável, consumo de processamento, consumo de memória.



3. Informações necessárias do TP 6:

3.1. Informações sobre as máquinas pertencentes à sub-rede do IP específico.

IP	porta	estado	protocolo	product
192.168.0.1	22	open	tcp	Dropbear sshd
192.168.0.1	23	open	tcp	BusyBox telnetd
192.168.0.1	80	open	tcp	
192.168.0.1	1900	open	tcp	Portable SDK for UPnP devices
192.168.0.3	8009	open	tcp	Amazon Whisperplay DIAL REST service
192.168.0.3	9080	open	tcp	
192.168.0.111	21	open	tcp	Brother/HP printer ftpd

3.2. Informações sobre as portas dos diferentes IPs obtidos nessa sub-rede.

192.168.0.3	9080	open	tcp	
192.168.0.111	21	open	tcp	Brother/HP printer ftpd
192.168.0.111	23	open	tcp	Brother/HP printer telnetd
192.168.0.111	80	open	tcp	Debut embedded httpd
192.168.0.111	443	open	tcp	Debut embedded httpd
192.168.0.111	515	open	tcp	
192.168.0.111	631	open	tcp	
192.168.0.111	9100	open	tcp	
192.168.0.112	135	open	tcp	Microsoft Windows RPC
192.168.0.112	139	open	tcp	Microsoft Windows netbios-ssn
192.168.0.112	445	open	tcp	
192.168.0.112	808	open	tcp	.NET Message Framing
192.168.0.112	1688	open	tcp	Microsoft Windows RPC
192.168.0.112	2869	open	tcp	Microsoft HTTPAPI httpd
192.168.0.112	5357	open	tcp	Microsoft HTTPAPI httpd

Interfaces:

4. Informações necessárias do TP 7:

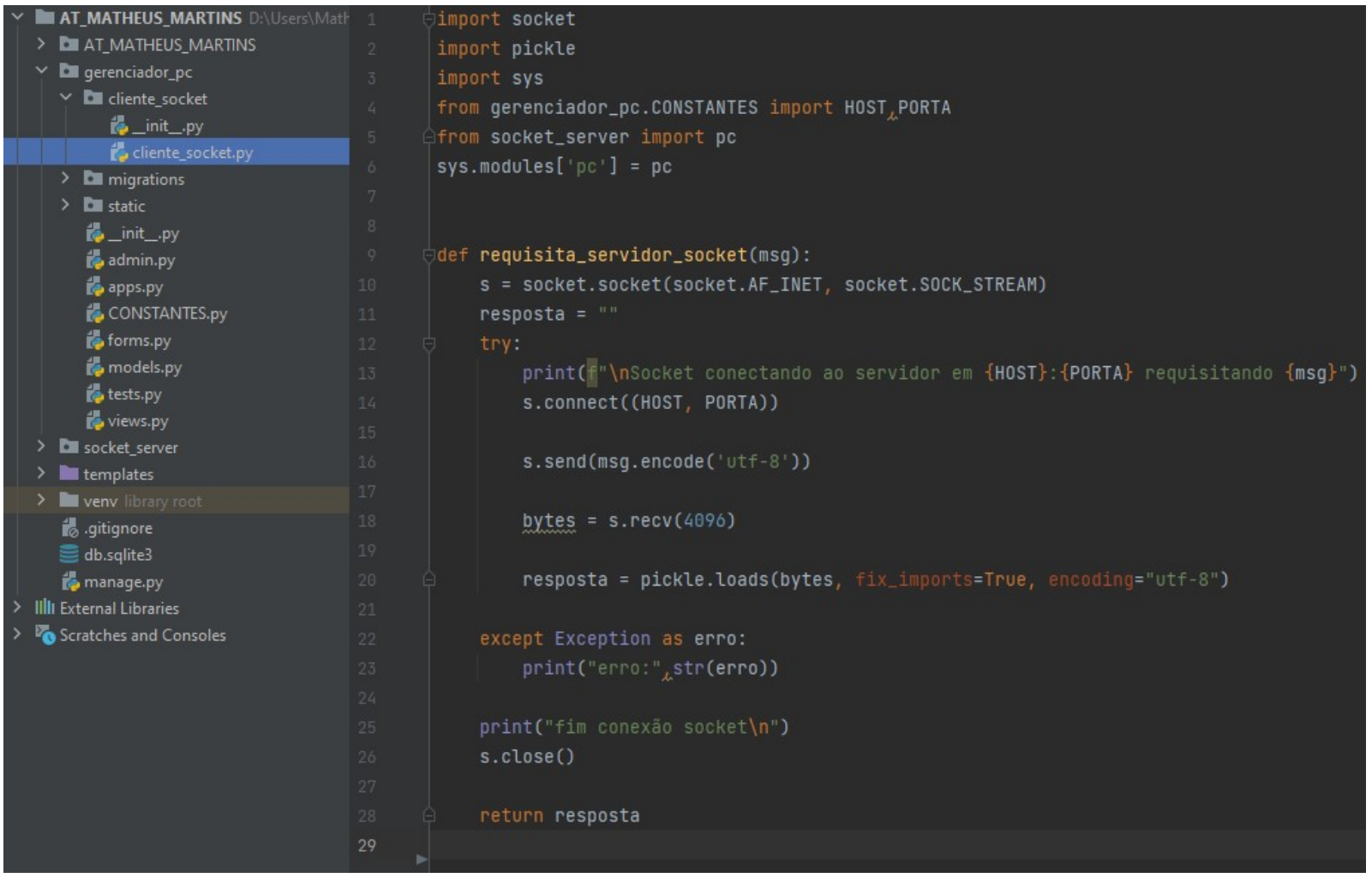
4.1. Ao menos 3 informações de interfaces de redes (exemplos: interfaces disponíveis, IP, gateway, máscara de subrede, etc.).

Interfaces:

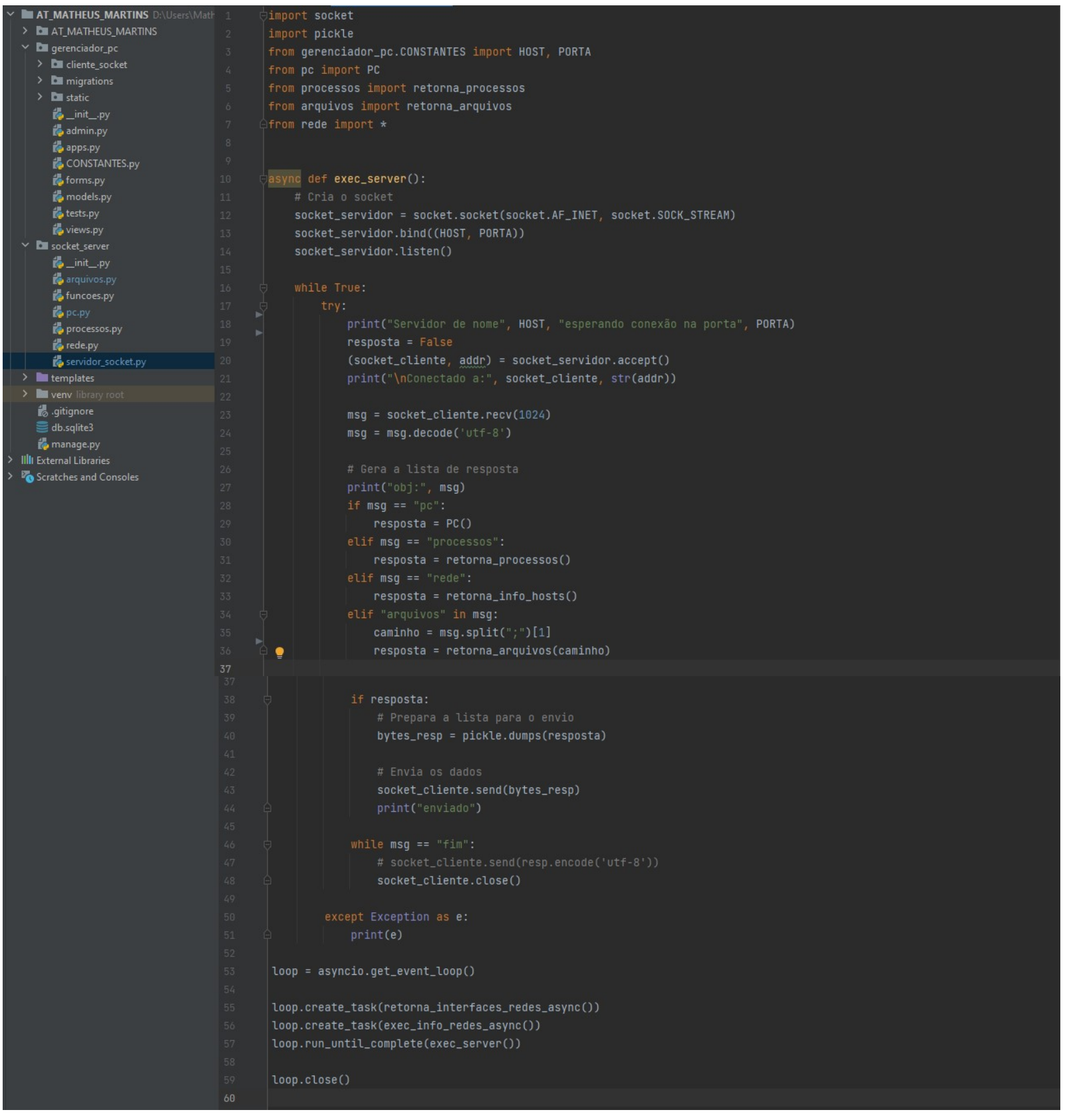
rede	familia	address	netmask	broadcast	ptp	ativa	duplex	speed	mtu	bytes_sent	bytes_recv	packets_sent	packets_recv	errin	errout	dropin	dropout
Ethernet	AddressFamily.AF_INET6	fe80::2c0c:2d25:68df:8f95				Sim	NIC_DUPLEX_FULL	100 Mb	1500 bytes	51.01 MB	39.81 MB	127.73 KB	126.96 KB	0.00 B	0.00 B	0.00 B	0.00 B
Loopback Pseudo-Interface 1	AddressFamily.AF_INET6	::1				Sim	NIC_DUPLEX_FULL	1073 Mb	1500 bytes	0.00 B	0.00 B	0.00 B	0.00 B	0.00 B	0.00 B	0.00 B	0.00 B

5. O TP8 corresponde à criação do cliente e servidor.

Cliente:


AT_MATHEUS_MARTINS D:\Users\Math
AT_MATHEUS_MARTINS
gerenciador_pc
cliente_socket
__init__.py
cliente_socket.py
migrations
static
__init__.py
admin.py
apps.py
CONSTANTES.py
forms.py
models.py
tests.py
views.py
socket_server
templates
venv library root
.gitignore
db.sqlite3
manage.py
External Libraries
Scratches and Consoles
import socket
import pickle
import sys
from gerenciador_pc.CONSTANTES import HOST, PORTA
from socket_server import pc
sys.modules['pc'] = pc
def requisita_servidor_socket(msg):
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
resposta = ""
try:
print(f"\nSocket conectando ao servidor em {HOST}:{PORTA} requisitando {msg}")
s.connect((HOST, PORTA))
s.send(msg.encode('utf-8'))
bytes = s.recv(4096)
resposta = pickle.loads(bytes, fix_imports=True, encoding="utf-8")
except Exception as erro:
print("erro:", str(erro))
print("fim conexão socket\n")
s.close()
return resposta

Servidor:


AT_MATHEUS_MARTINS D:\Users\Math
AT_MATHEUS_MARTINS
gerenciador_pc
cliente_socket
migrations
static
__init__.py
admin.py
apps.py
CONSTANTES.py
forms.py
models.py
tests.py
views.py
socket_server
__init__.py
arquivos.py
funcoes.py
pc.py
processos.py
rede.py
servidor_socket.py
templates
venv library root
.gitignore
db.sqlite3
manage.py
External Libraries
Scratches and Consoles
import socket
import pickle
from gerenciador_pc.CONSTANTES import HOST, PORTA
from pc import PC
from processos import retorna_processos
from arquivos import retorna_arquivos
from rede import *
async def exec_server():
# Cria o socket
socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket_servidor.bind((HOST, PORTA))
socket_servidor.listen()
while True:
try:
print("Servidor de nome", HOST, "esperando conexão na porta", PORTA)
resposta = False
(socket_cliente, addr) = socket_servidor.accept()
print("\nConectado a:", socket_cliente, str(addr))
msg = socket_cliente.recv(1024)
msg = msg.decode('utf-8')
# Gera a lista de resposta
print("obj:", msg)
if msg == "pc":
resposta = PC()
elif msg == "processos":
resposta = retorna_processos()
elif msg == "rede":
resposta = retorna_info_hosts()
elif "arquivos" in msg:
caminho = msg.split(";")[1]
resposta = retorna_arquivos(caminho)
if resposta:
# Prepara a lista para o envio
bytes_resp = pickle.dumps(resposta)
# Envia os dados
socket_cliente.send(bytes_resp)
print("enviado")
while msg == "fim":
# socket_cliente.send(resp.encode('utf-8'))
socket_cliente.close()
except Exception as e:
print(e)
loop = asyncio.get_event_loop()
loop.create_task(retorna_interfaces_redes_async())
loop.create_task(exec_info_redes_async())
loop.run_until_complete(exec_server())
loop.close()