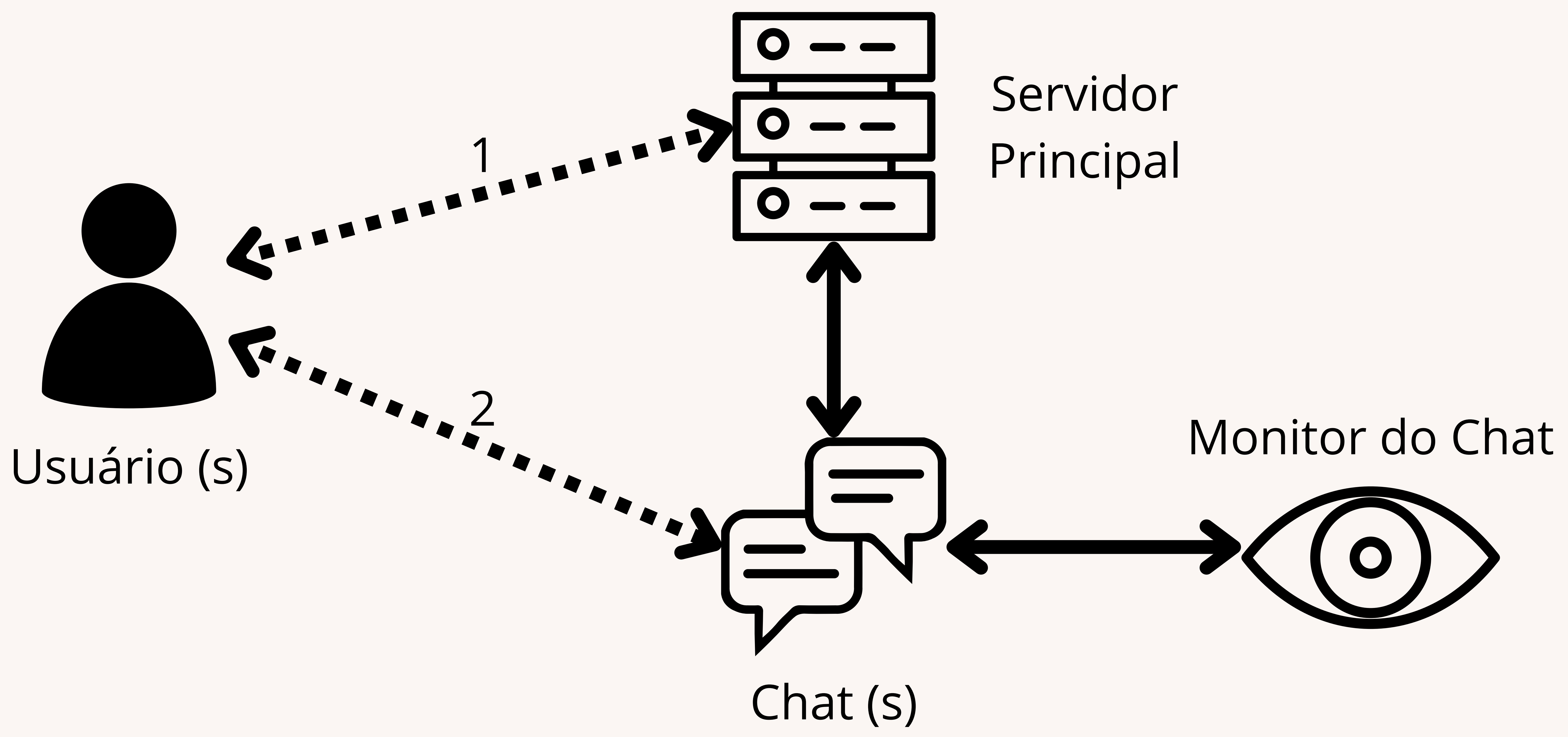
CHATS SEM PERSISTÊNCIA

INE5418-T1

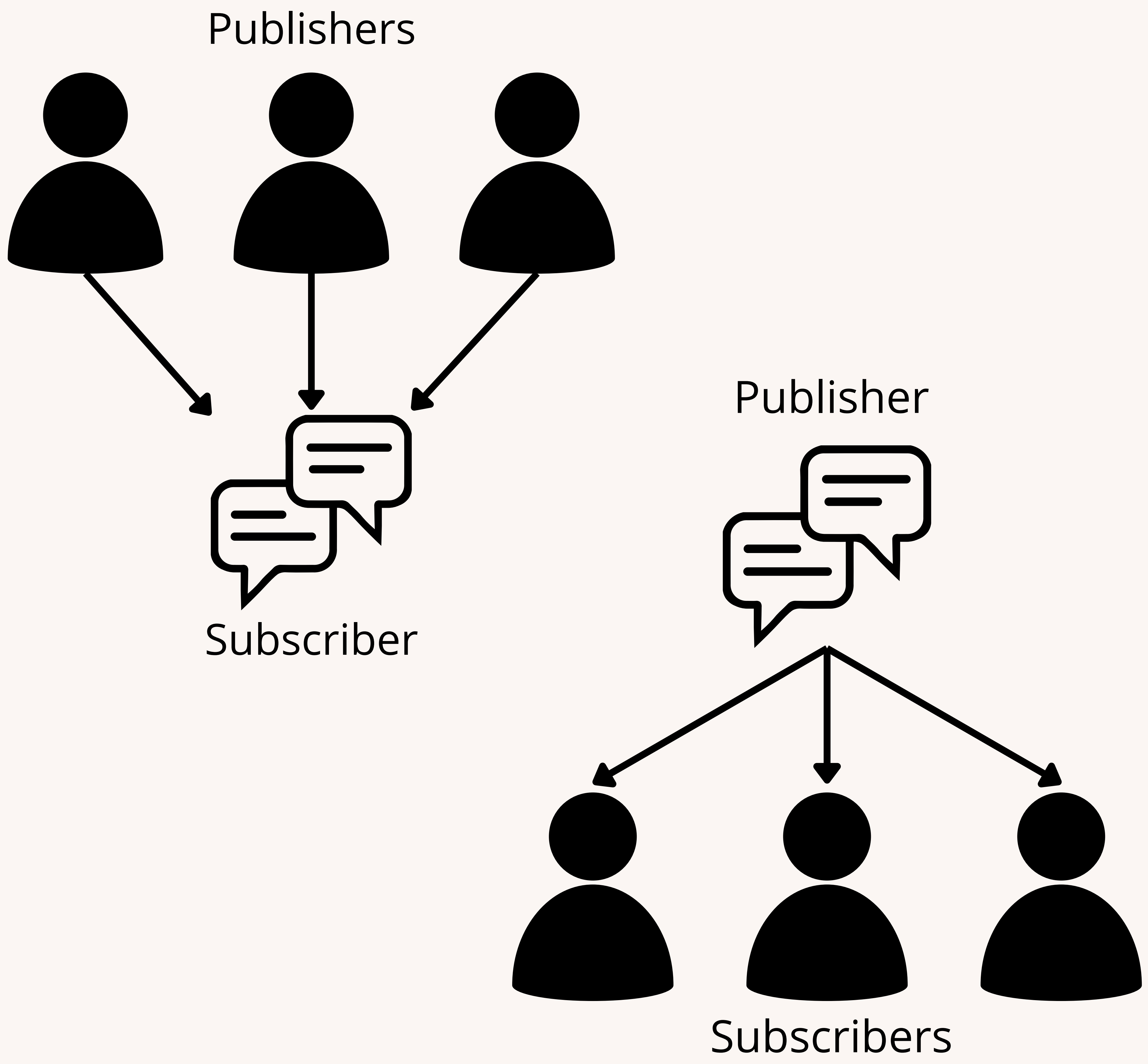
Alunos: Guilherme Adenilson de Jesus (22100620)
Vicente Cardoso dos Santos (22102203)

Componentes do Sistema



Chat

- **Publisher-Subscriber**
- Duas ZMQ (PUB e SUB)
- Transmite a mensagem recebida para todos os usuários do chat



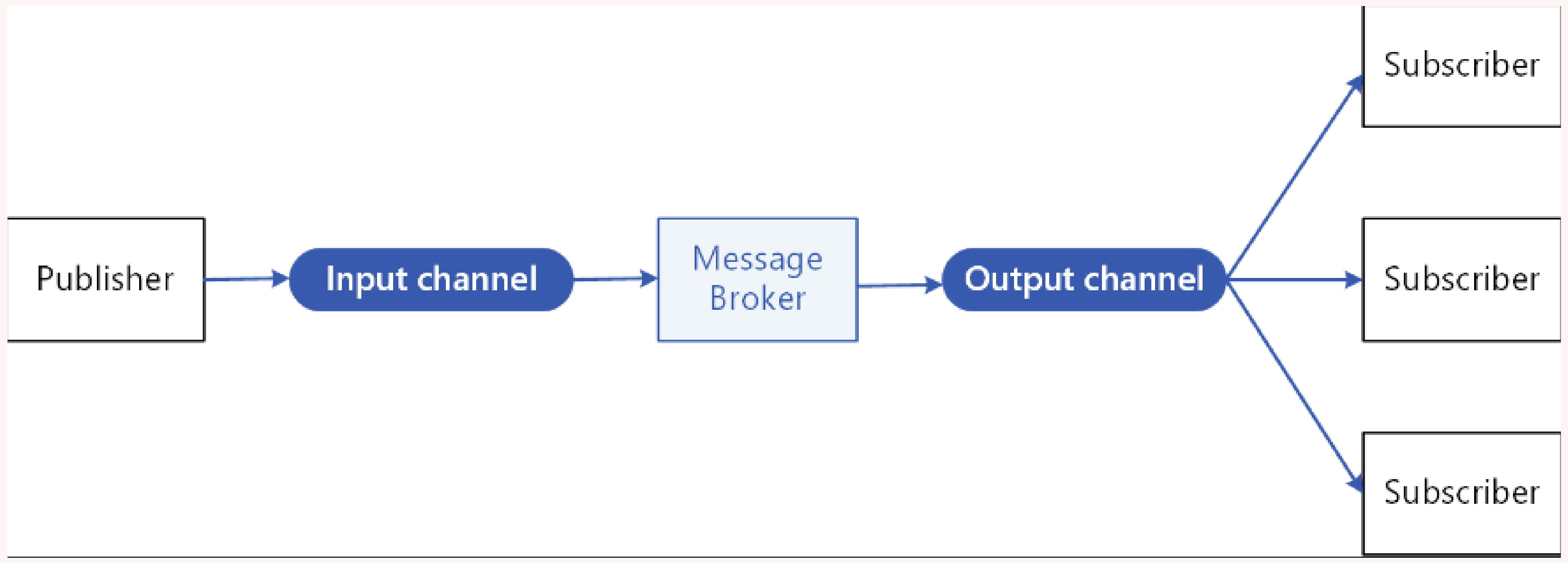
Publisher-Subscriber

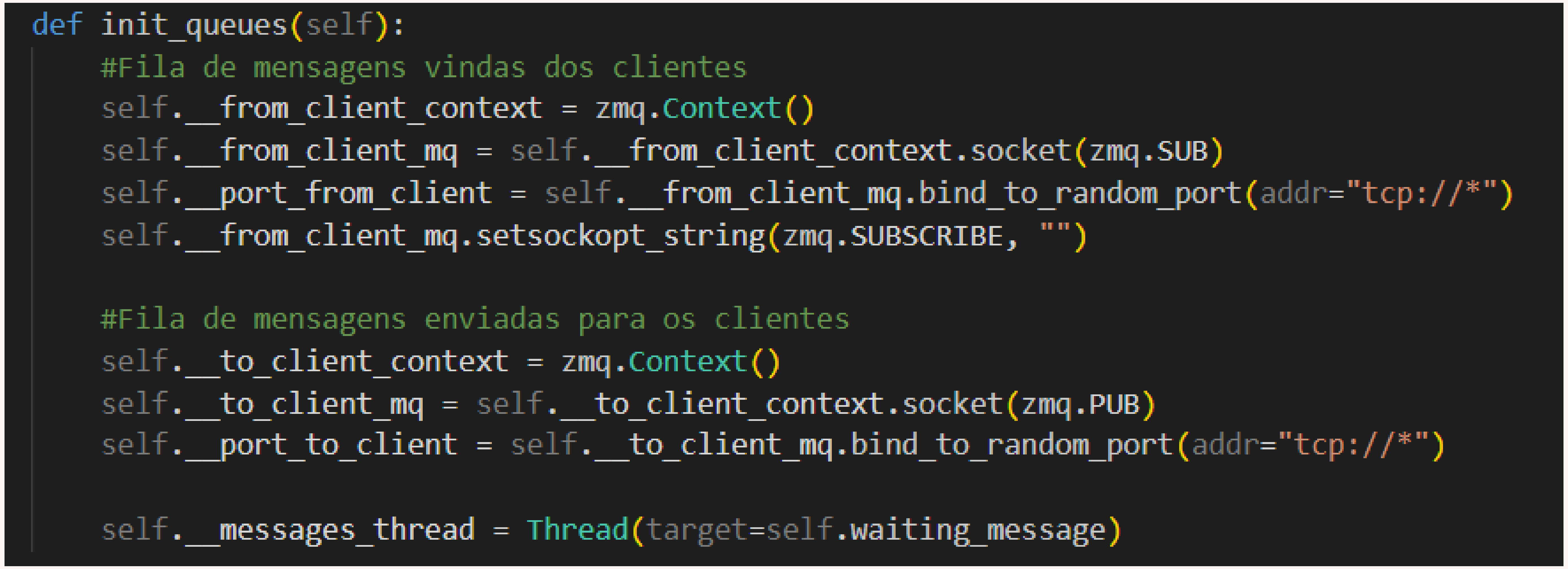
- Subsistema assíncrono de mensagens

- Publishers: canal de entrada

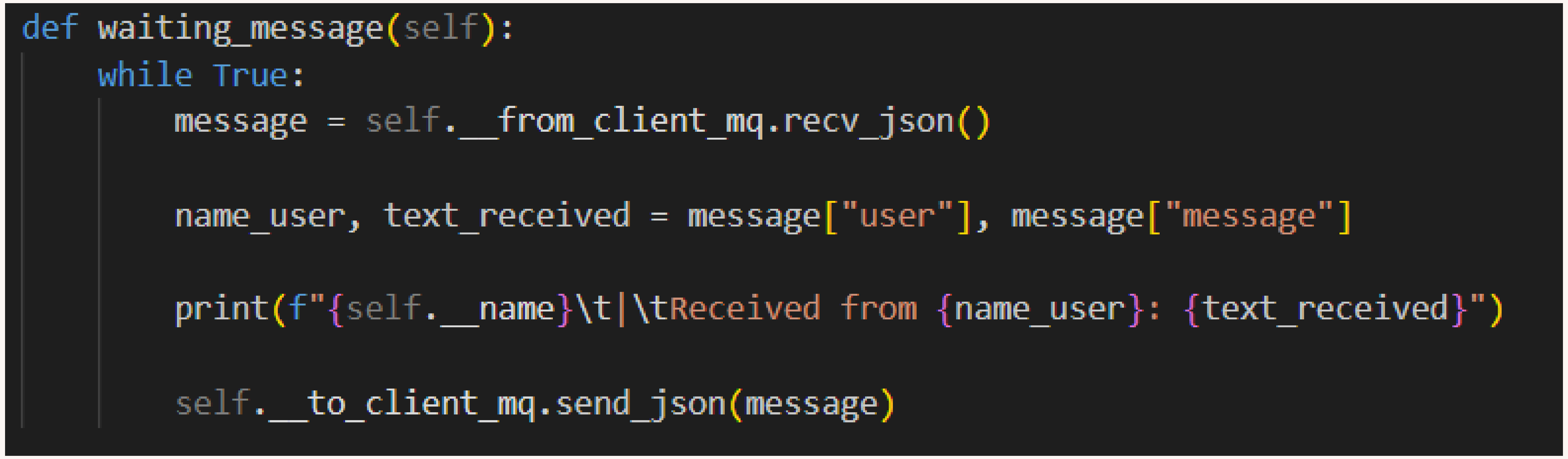
- Subscribers: canal de saída

- A mensagem na saída é replicada para todos os subscribers





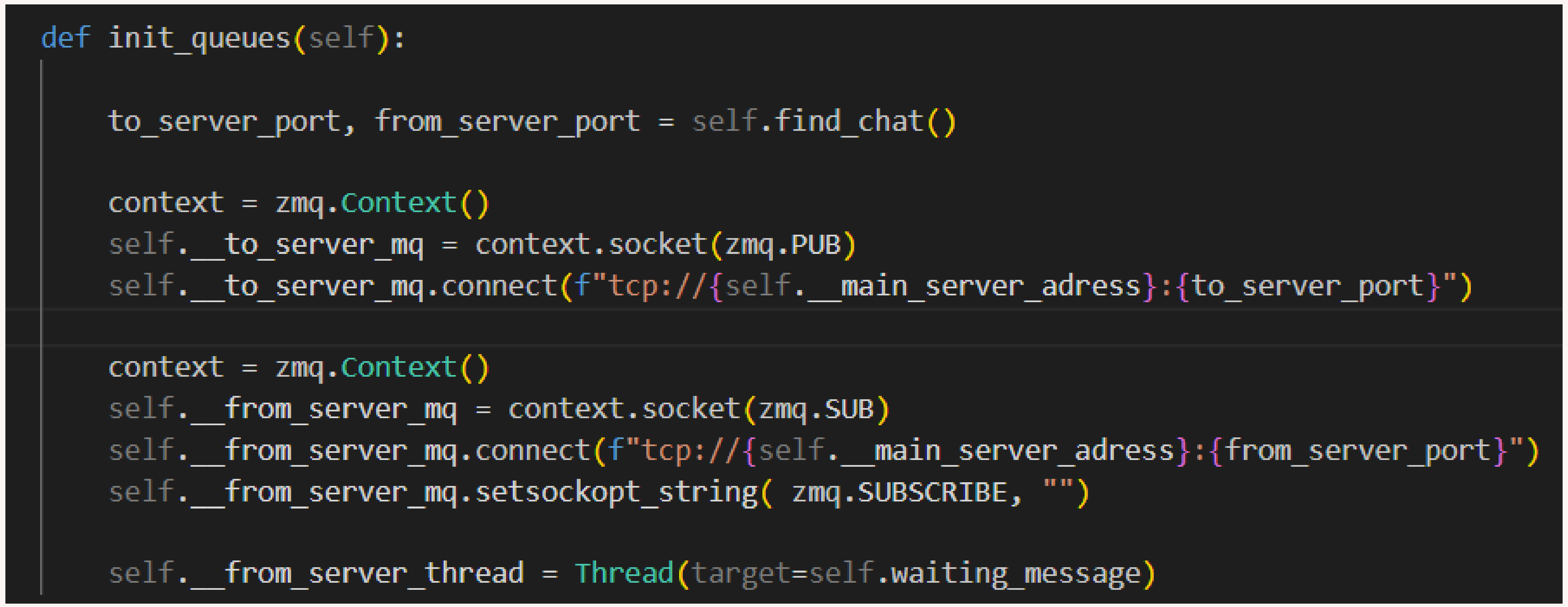
```python
def init_queues(self):
    #Fila de mensagens vindas dos clientes
    self.__from_client_context = zmq.Context()
    self.__from_client_mq = self.__from_client_context.socket(zmq.SUB)
    self.__port_from_client = self.__from_client_mq.bind_to_random_port(addr="tcp://*")
    self.__from_client_mq.setsockopt_string(zmq.SUBSCRIBE, "")

    #Fila de mensagens enviadas para os clientes
    self.__to_client_context = zmq.Context()
    self.__to_client_mq = self.__to_client_context.socket(zmq.PUB)
    self.__port_to_client = self.__to_client_mq.bind_to_random_port(addr="tcp://*")

    self.__messages_thread = Thread(target=self.waiting_message)
```



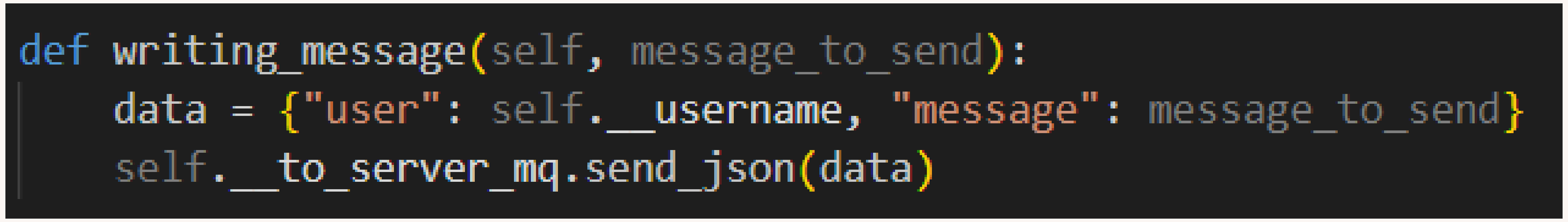
```python
def waiting_message(self):
    while True:
        message = self.__from_client_mq.recv_json()

        name_user, text_received = message["user"], message["message"]

        print(f"{self.__name}\t|\tReceived from {name_user}: {text_received}")

        self.__to_client_mq.send_json(message)
```

No lado do Chat

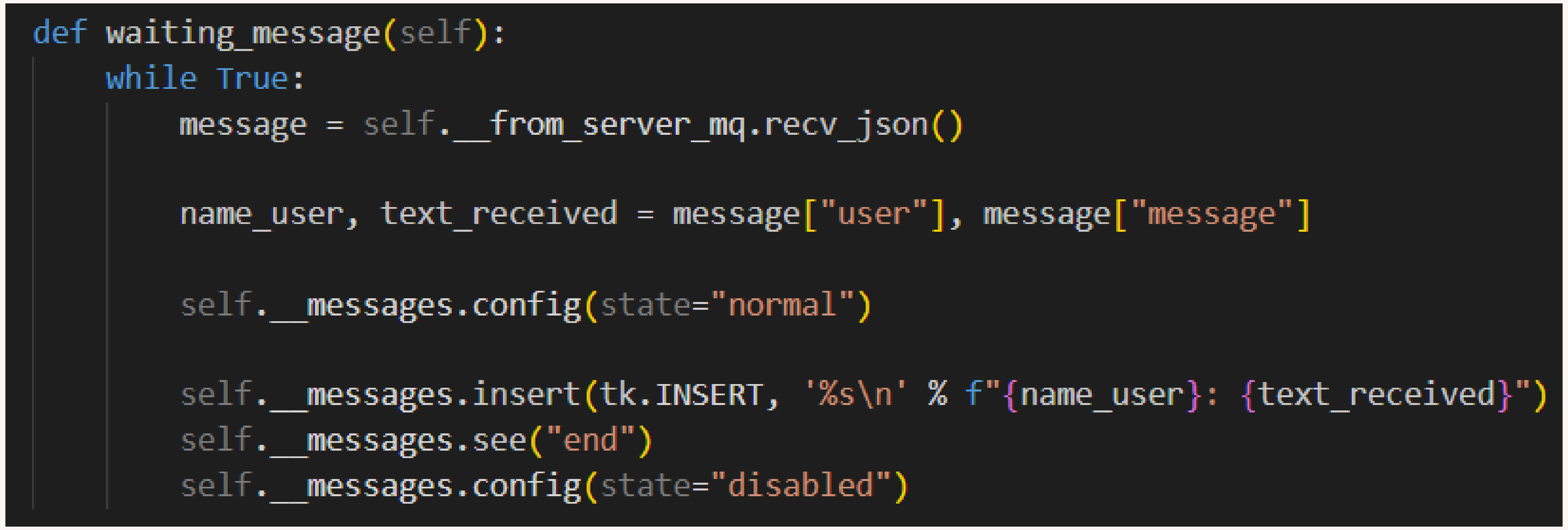
No lado do Usuário



```python
def init_queues(self):

    to_server_port, from_server_port = self.find_chat()

    context = zmq.Context()
    self.__to_server_mq = context.socket(zmq.PUB)
    self.__to_server_mq.connect(f"tcp://{self.__main_server_adress}:{to_server_port}")

    context = zmq.Context()
    self.__from_server_mq = context.socket(zmq.SUB)
    self.__from_server_mq.connect(f"tcp://{self.__main_server_adress}:{from_server_port}")
    self.__from_server_mq.setsockopt_string( zmq.SUBSCRIBE, "")

    self.__from_server_thread = Thread(target=self.waiting_message)
```



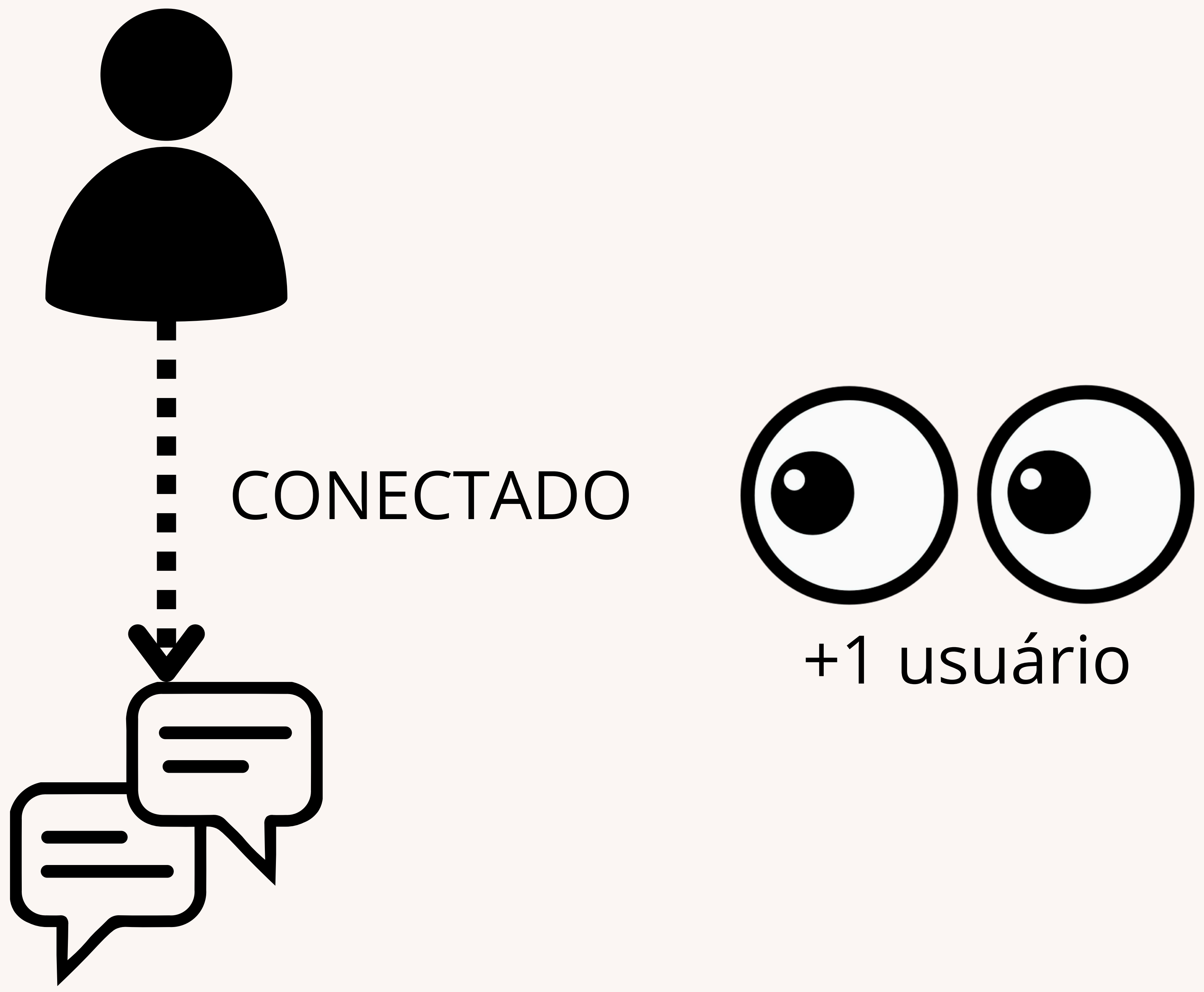
```python
def writing_message(self, message_to_send):
    data = {"user": self.__username, "message": message_to_send}
    self.__to_server_mq.send_json(data)
```

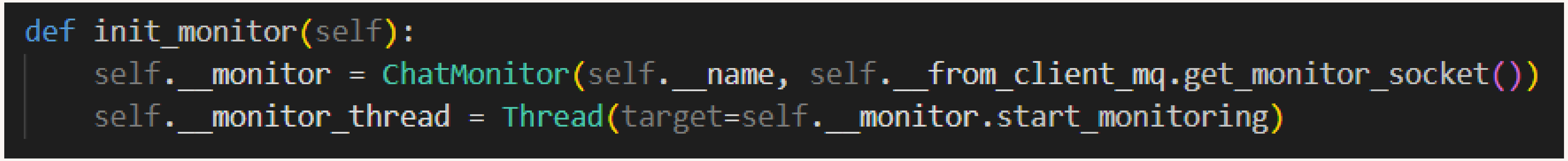


```python
def waiting_message(self):
    while True:
        message = self.__from_server_mq.recv_json()

        name_user, text_received = message["user"], message["message"]

        self.__messages.config(state="normal")

        self.__messages.insert(tk.INSERT, '%s\n' % f"{name_user}: {text_received}")
        self.__messages.see("end")
        self.__messages.config(state="disabled")
```

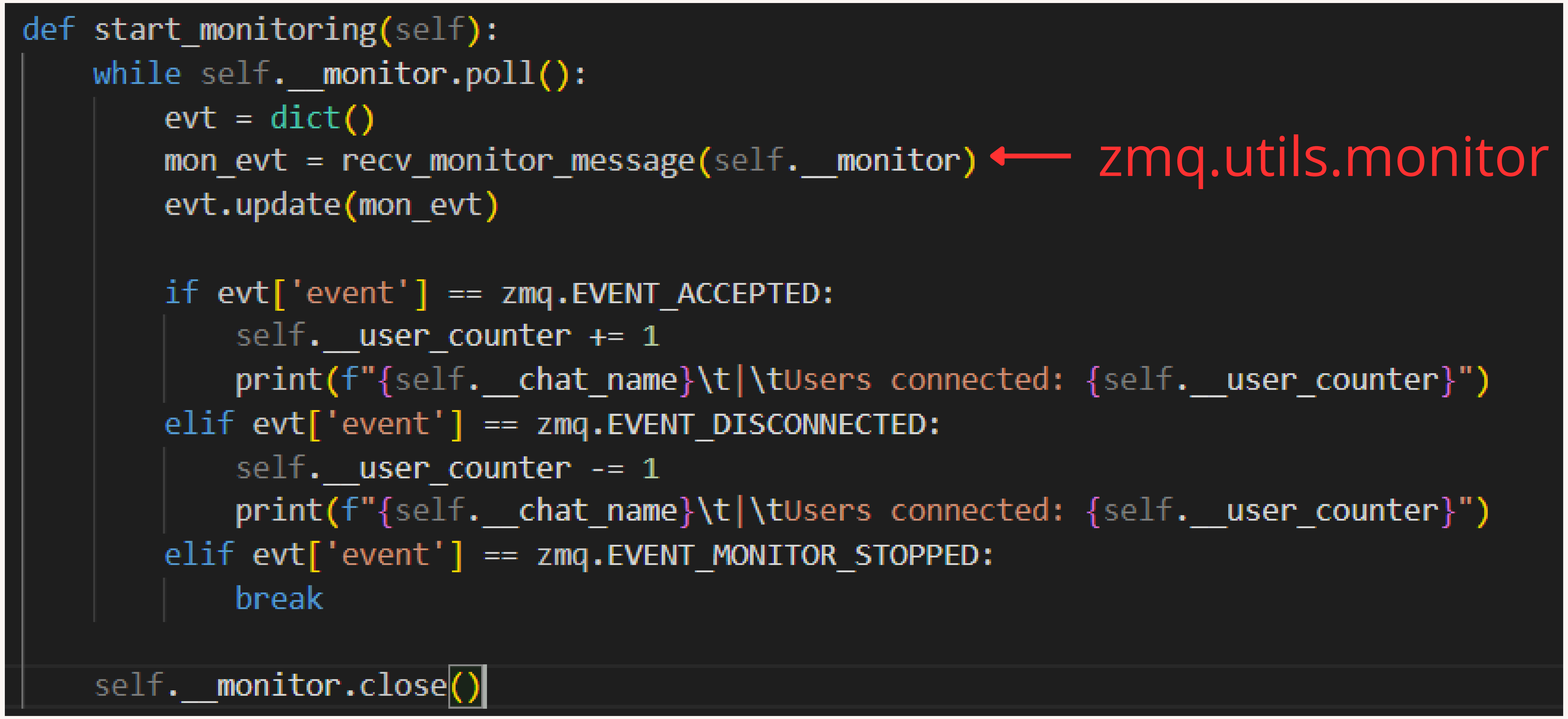
Monitor do Chat

- Contabiliza o número de usuários conectados
- Utiliza a detecção de eventos do zmq





```python
def init_monitor(self):
    self.__monitor = ChatMonitor(self.__name, self.__from_client_mq.get_monitor_socket())
    self.__monitor_thread = Thread(target=self.__monitor.start_monitoring)
```



```python
def start_monitoring(self):
    while self.__monitor.poll():
        evt = dict()
        mon_evt = recv_monitor_message(self.__monitor)
        evt.update(mon_evt)

        if evt['event'] == zmq.EVENT_ACCEPTED:
            self.__user_counter += 1
            print(f"{self.__chat_name}\t|\tUsers connected: {self.__user_counter}")
        elif evt['event'] == zmq.EVENT_DISCONNECTED:
            self.__user_counter -= 1
            print(f"{self.__chat_name}\t|\tUsers connected: {self.__user_counter}")
        elif evt['event'] == zmq.EVENT_MONITOR_STOPPED:
            break

    self.__monitor.close()
```

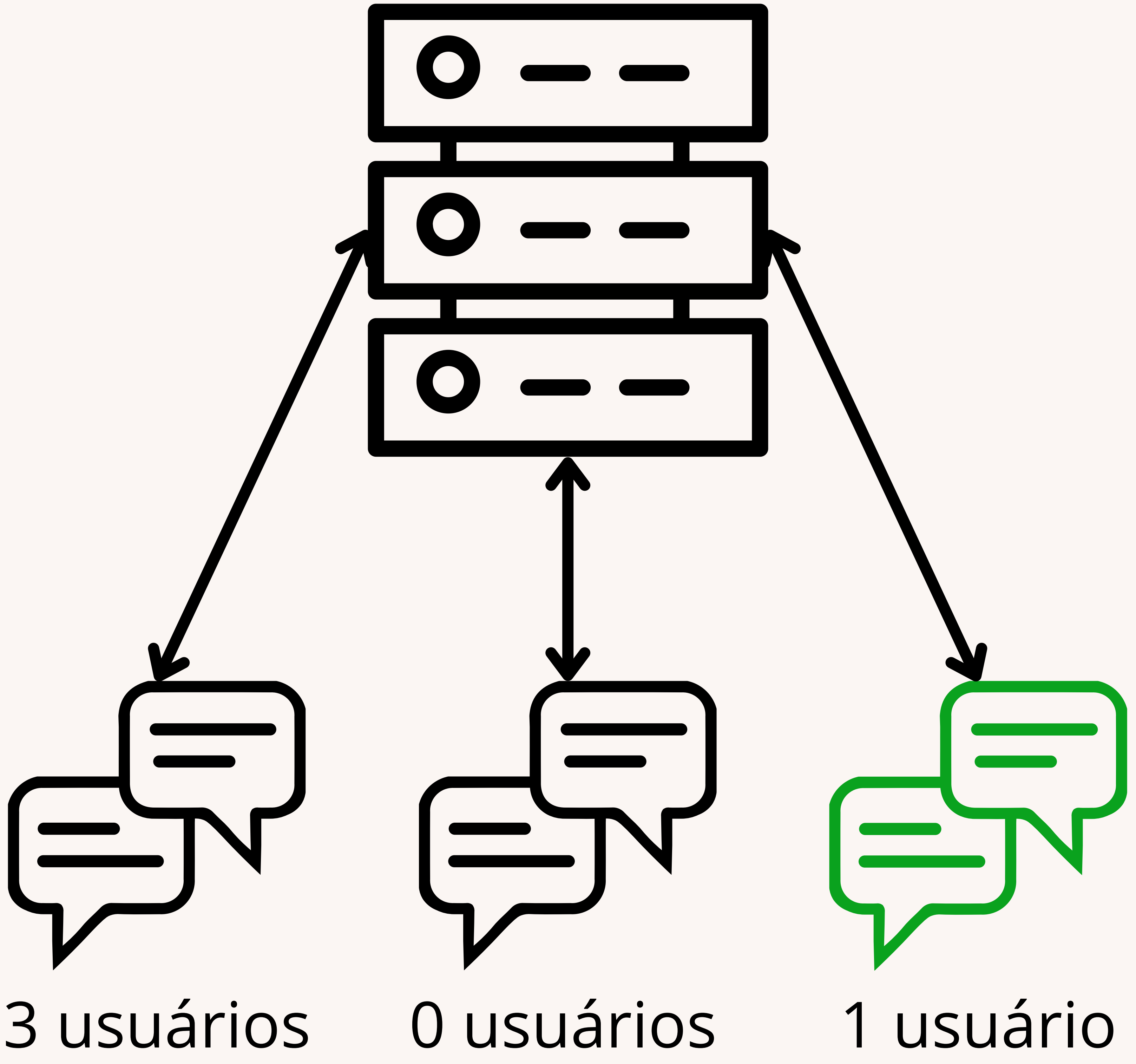
$\longleftarrow$ zmq.utils.monitor

Servidor Principal

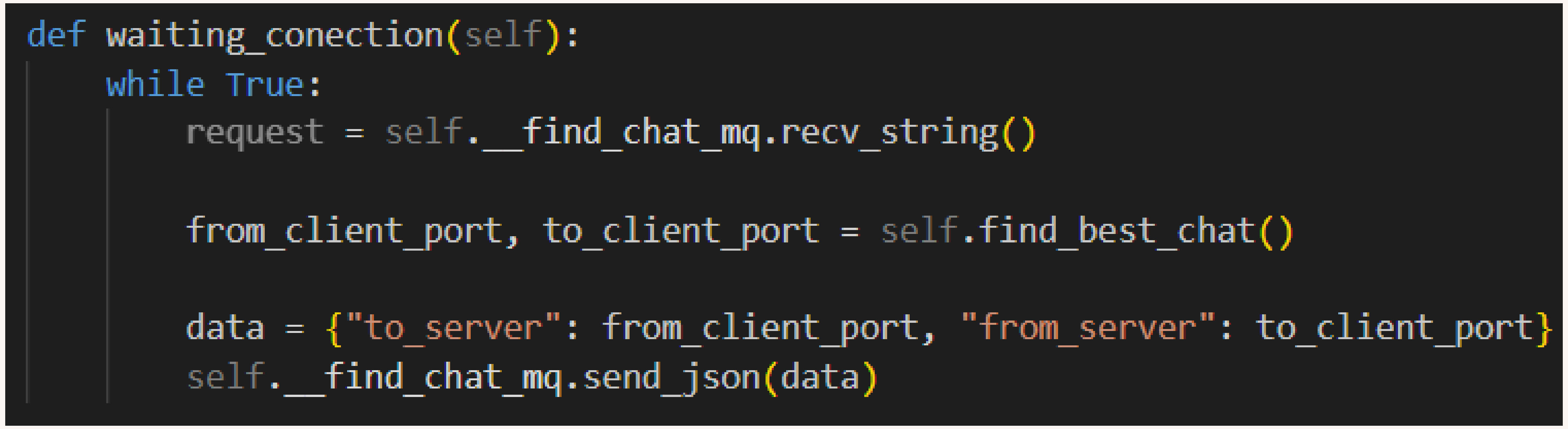
- **Load Balancer**
  - 1: Prioriza chat com 1 usuário
  - 2: Chat com menos usuários
  - 3: Se todos lotados, cria um novo

- Porta de entrada do usuário

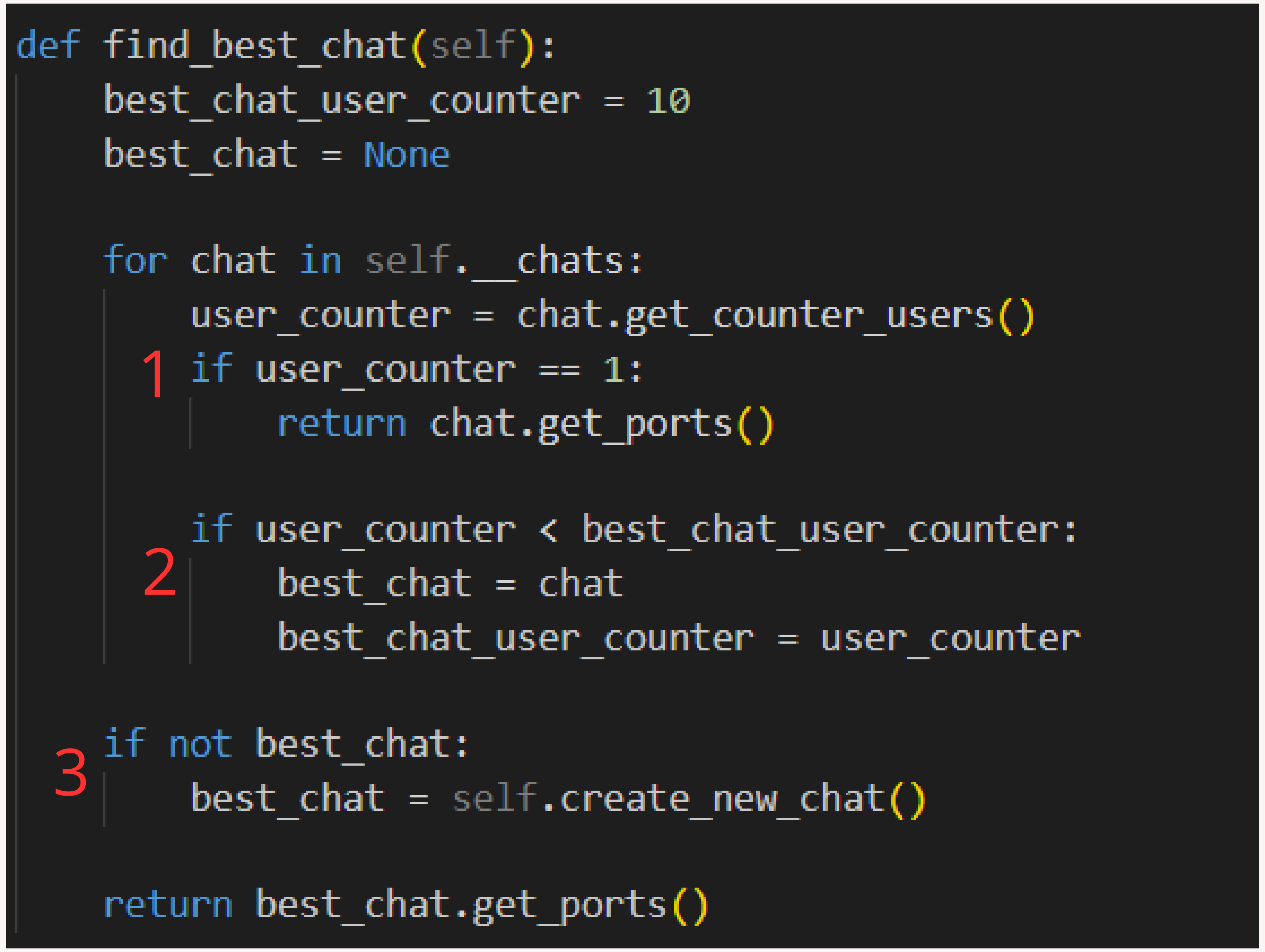
- **ZMQ REQ-REP:** Usuário requisita um chat, o servidor retorna as portas do chat escolhido pelo **load balancer**



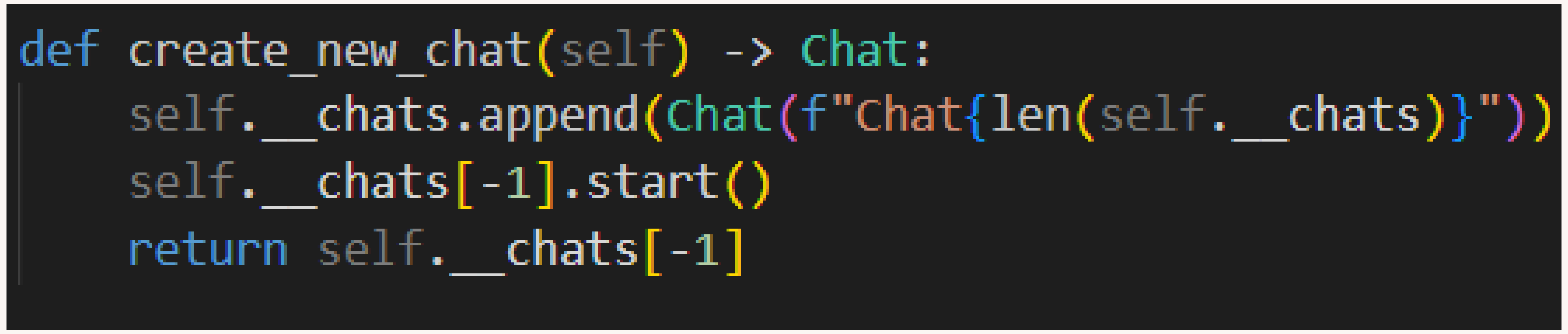
Escolhendo melhor chat



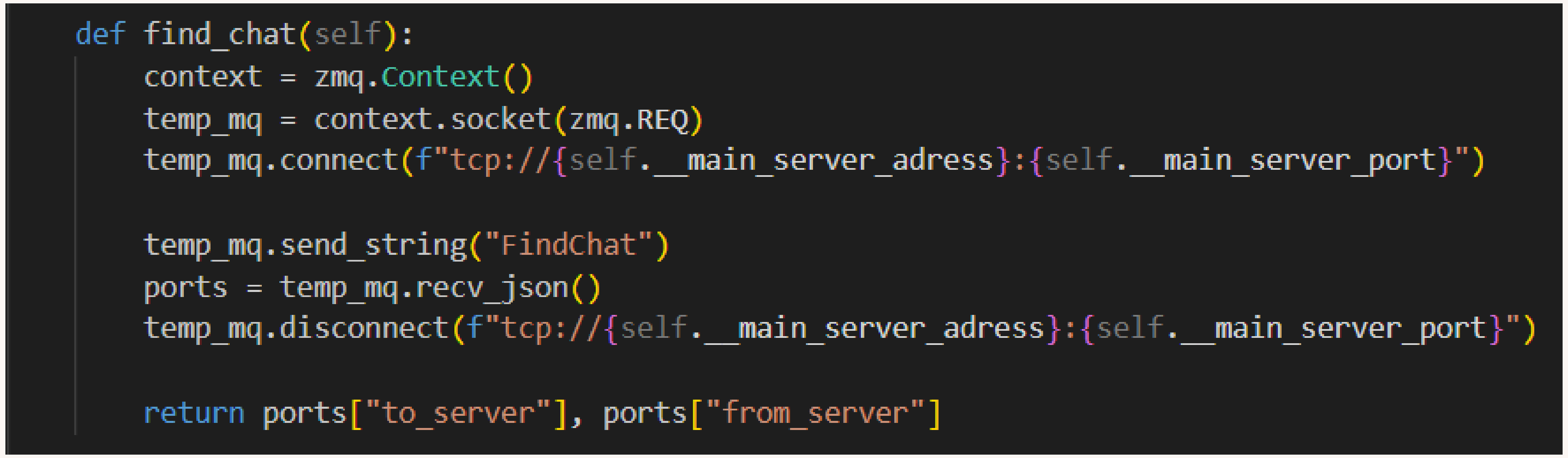
```python
def waiting_conection(self):
    while True:
        request = self.__find_chat_mq.recv_string()

        from_client_port, to_client_port = self.find_best_chat()

        data = {"to_server": from_client_port, "from_server": to_client_port}
        self.__find_chat_mq.send_json(data)
```



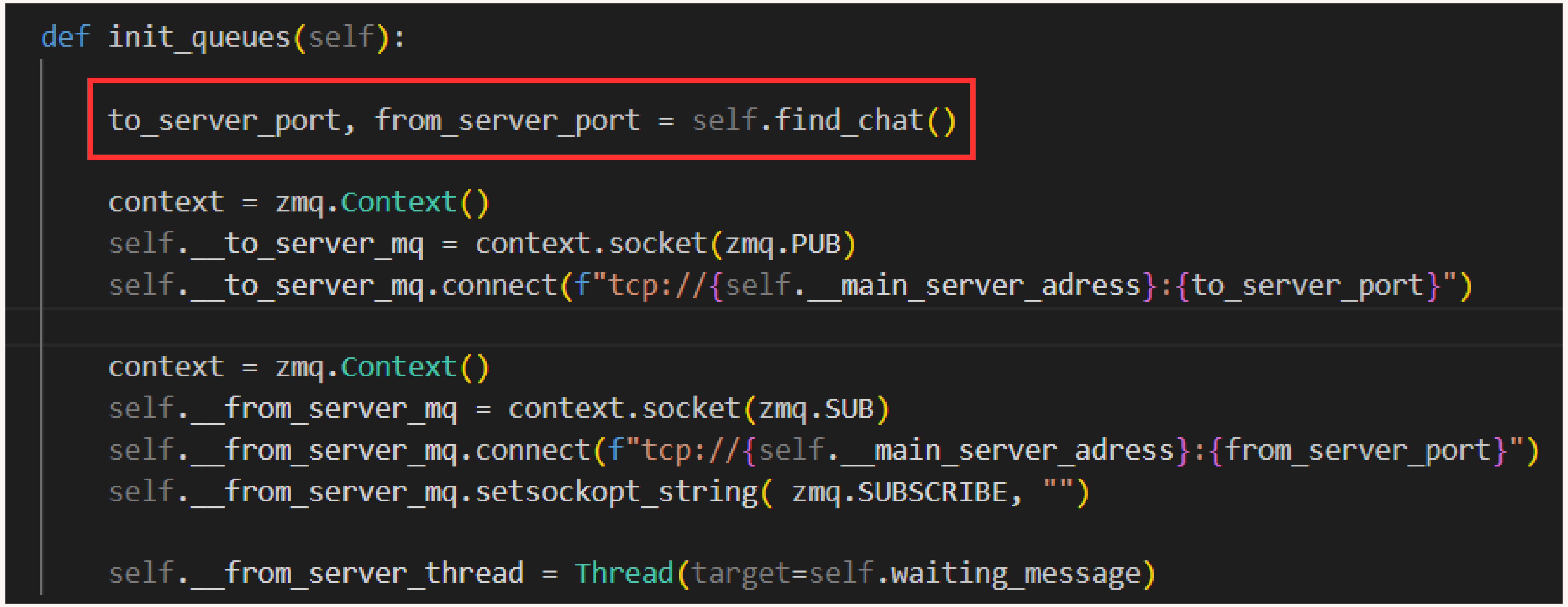
```python
def find_best_chat(self):
    best_chat_user_counter = 10
    best_chat = None

    for chat in self.__chats:
        user_counter = chat.get_counter_users()
      1 if user_counter == 1:
            return chat.get_ports()

        if user_counter < best_chat_user_counter:
      2     best_chat = chat
            best_chat_user_counter = user_counter

    if not best_chat:
  3     best_chat = self.create_new_chat()

    return best_chat.get_ports()
```



```python
def create_new_chat(self) -> Chat:
    self.__chats.append(Chat(f"Chat{len(self.__chats)}"))
    self.__chats[-1].start()
    return self.__chats[-1]
```



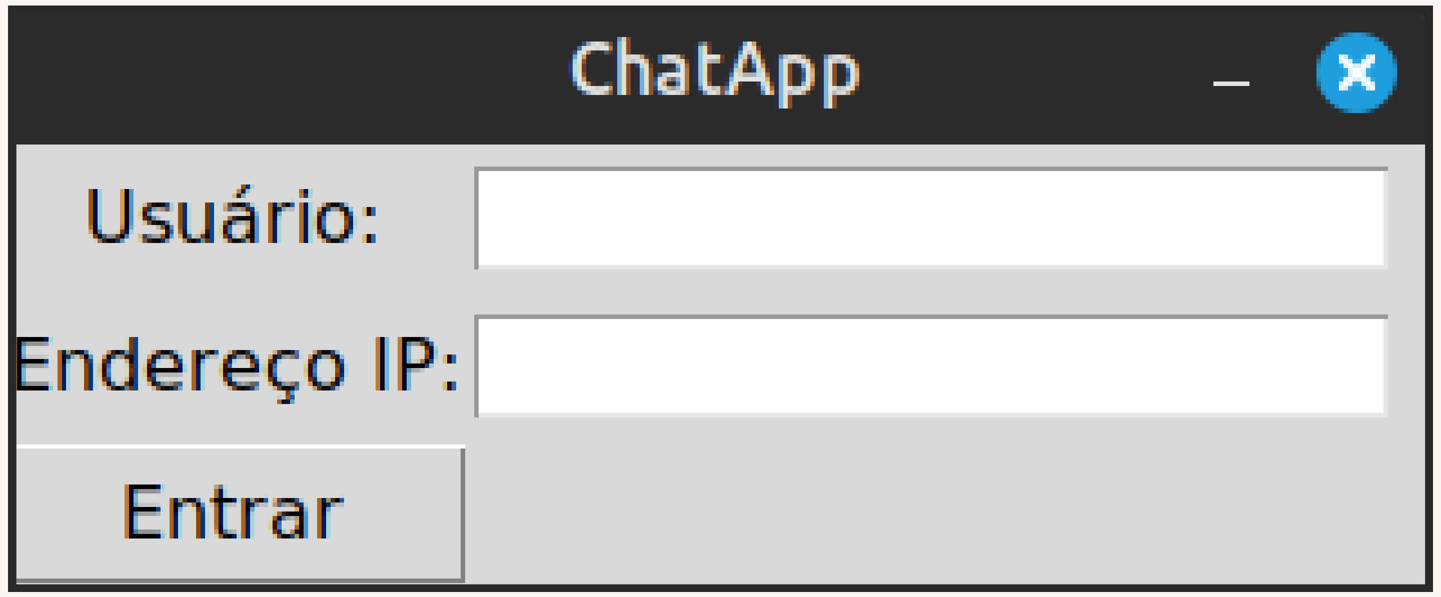
```python
    def find_chat(self):
        context = zmq.Context()
        temp_mq = context.socket(zmq.REQ)
        temp_mq.connect(f"tcp://{self.__main_server_adress}:{self.__main_server_port}")

        temp_mq.send_string("FindChat")
        ports = temp_mq.recv_json()
        temp_mq.disconnect(f"tcp://{self.__main_server_adress}:{self.__main_server_port}")

        return ports["to_server"], ports["from_server"]
```



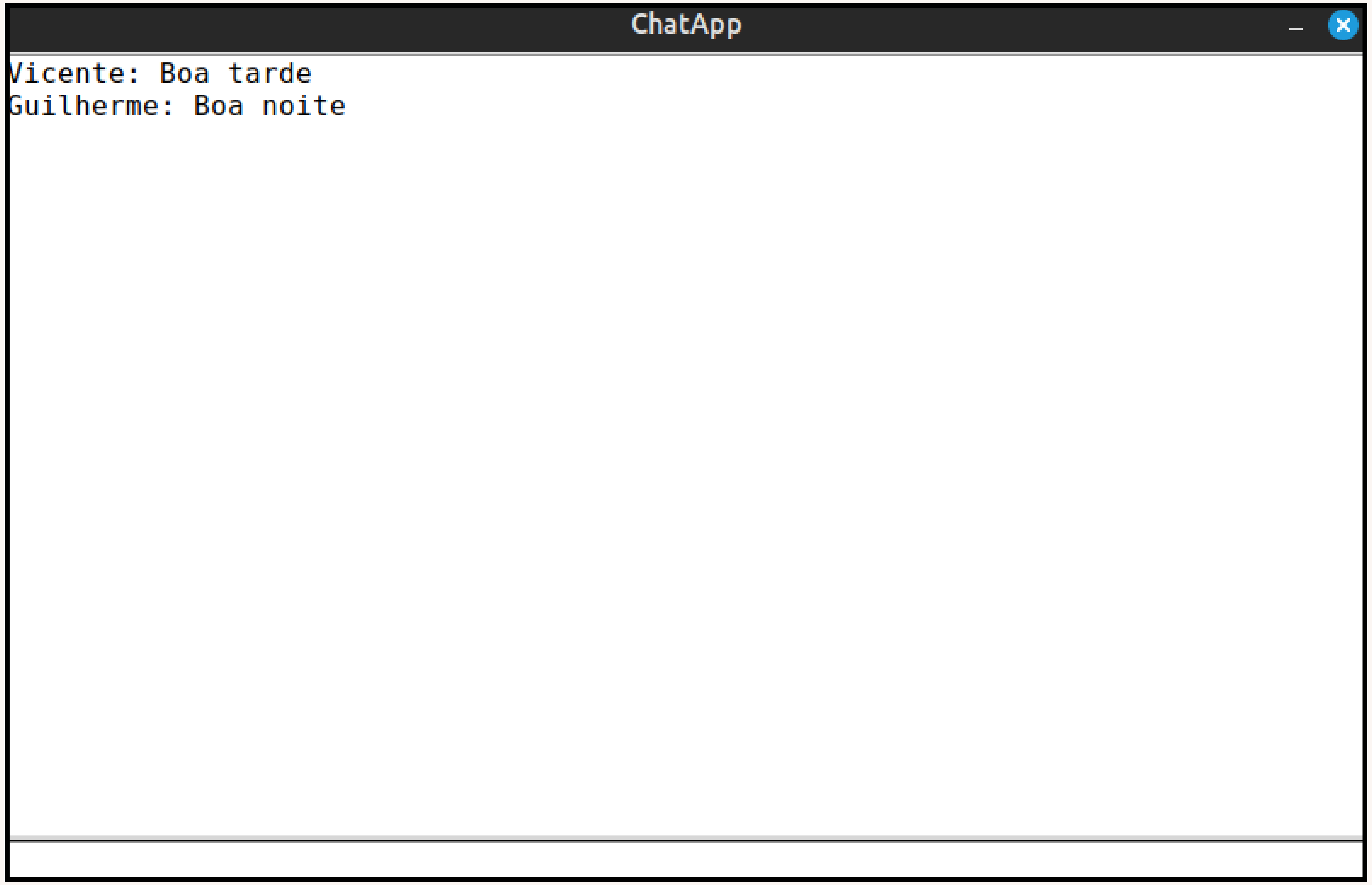
```python
def init_queues(self):

    to_server_port, from_server_port = self.find_chat()

    context = zmq.Context()
    self.__to_server_mq = context.socket(zmq.PUB)
    self.__to_server_mq.connect(f"tcp://{self.__main_server_adress}:{to_server_port}")

    context = zmq.Context()
    self.__from_server_mq = context.socket(zmq.SUB)
    self.__from_server_mq.connect(f"tcp://{self.__main_server_adress}:{from_server_port}")
    self.__from_server_mq.setsockopt_string( zmq.SUBSCRIBE, "")

    self.__from_server_thread = Thread(target=self.waiting_message)
```

No lado do usuário

Exemplo de funcionamento

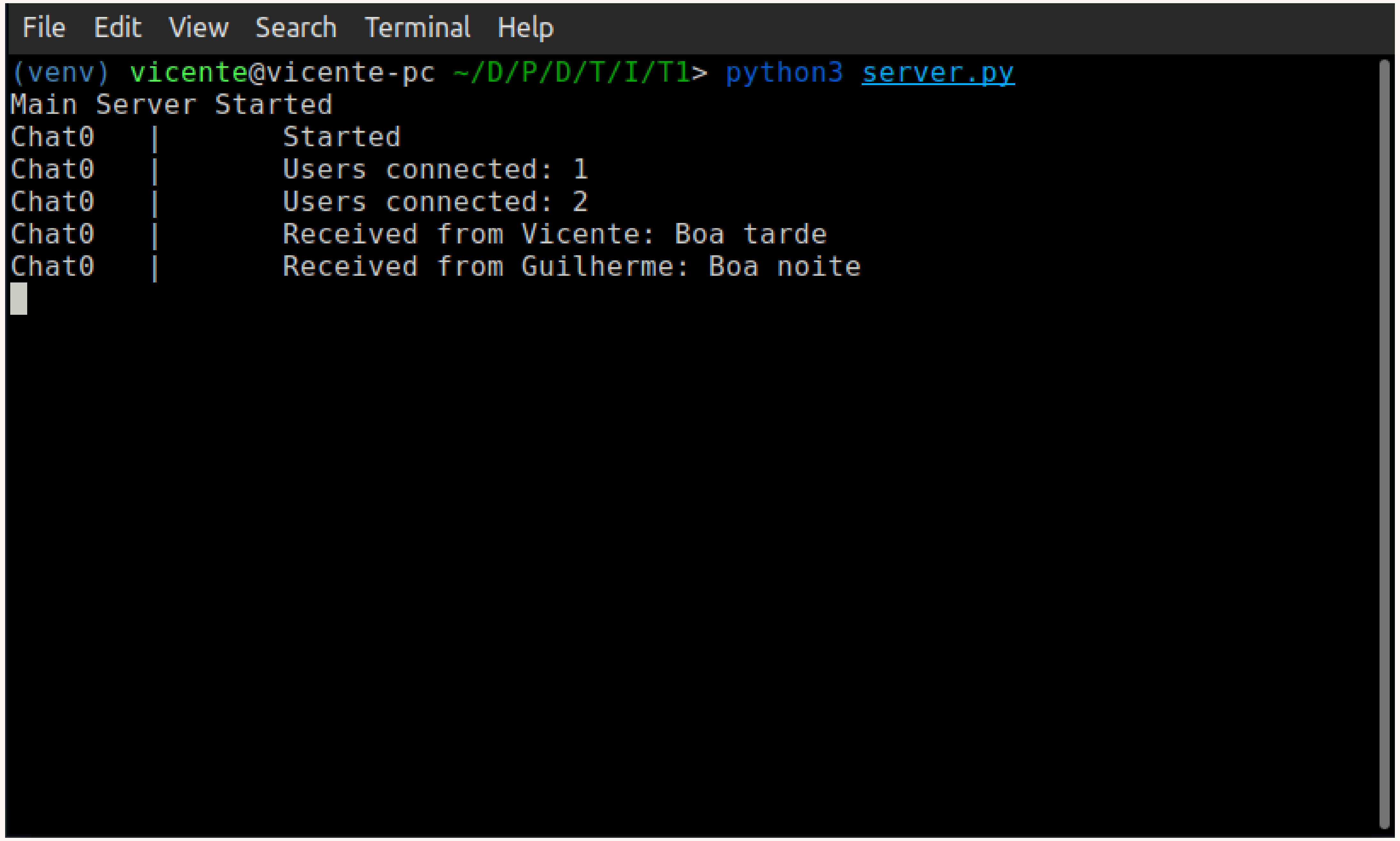


Endereço IP: em caso de usuário e servidor em mesma máquina, pode ser localhost ou 127.0.0.1



```
Vicente: Boa tarde
Guilherme: Boa noite
```

Usuários recebem mensagens posteriores à sua entrada



```
(venv) vicente@vicente-pc ~/D/P/D/T/I/T1> python3 server.py
Main Server Started
Chat0   |       Started
Chat0   |       Users connected: 1
Chat0   |       Users connected: 2
Chat0   |       Received from Vicente: Boa tarde
Chat0   |       Received from Guilherme: Boa noite
```

Resumo

Tecnologia utilizada: Fila de Mensagens (biblioteca zmq)

Padrões usados: Publisher-Subscriber e Load Balancer

Linguagem: Python3

Interface de usuário: Tkinter