IFES - Serra

Redes de Computadores
Cristina Klippel Dominicini

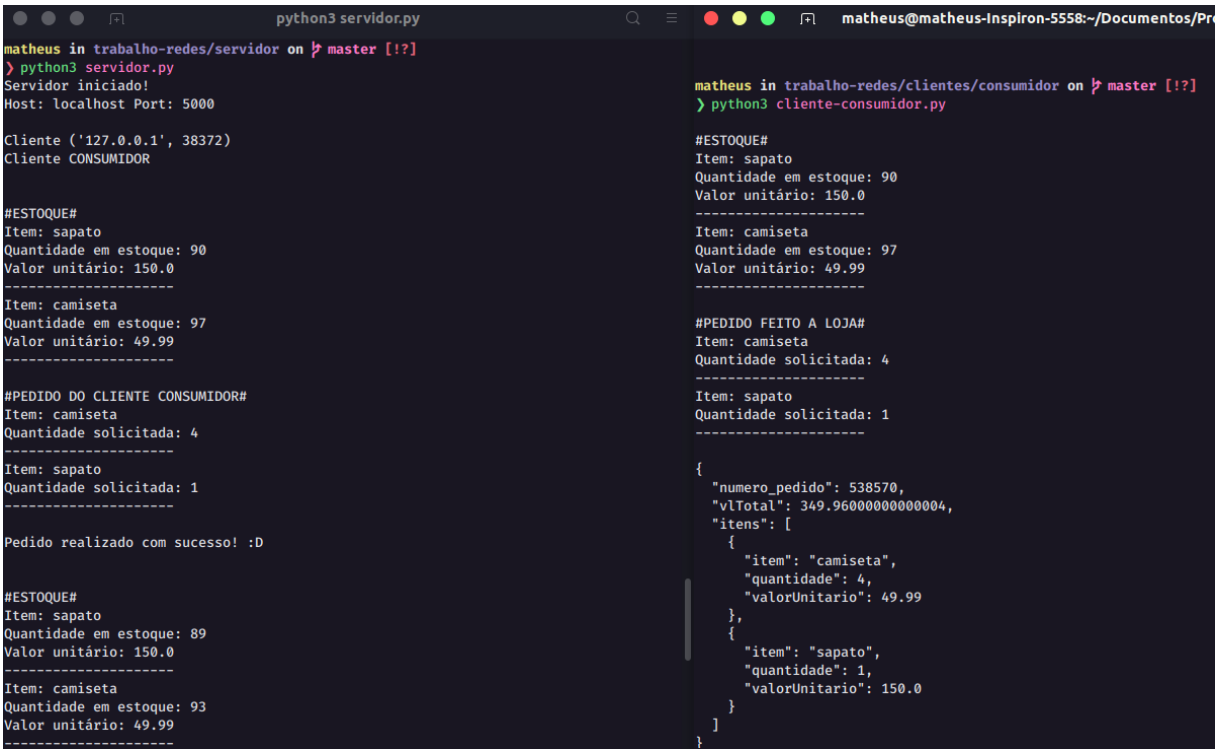
Matheus de Souza Pereira de Oliveira
20191BSI0301

Relatório do Trabalho #1
Utilizando API de socket em Python

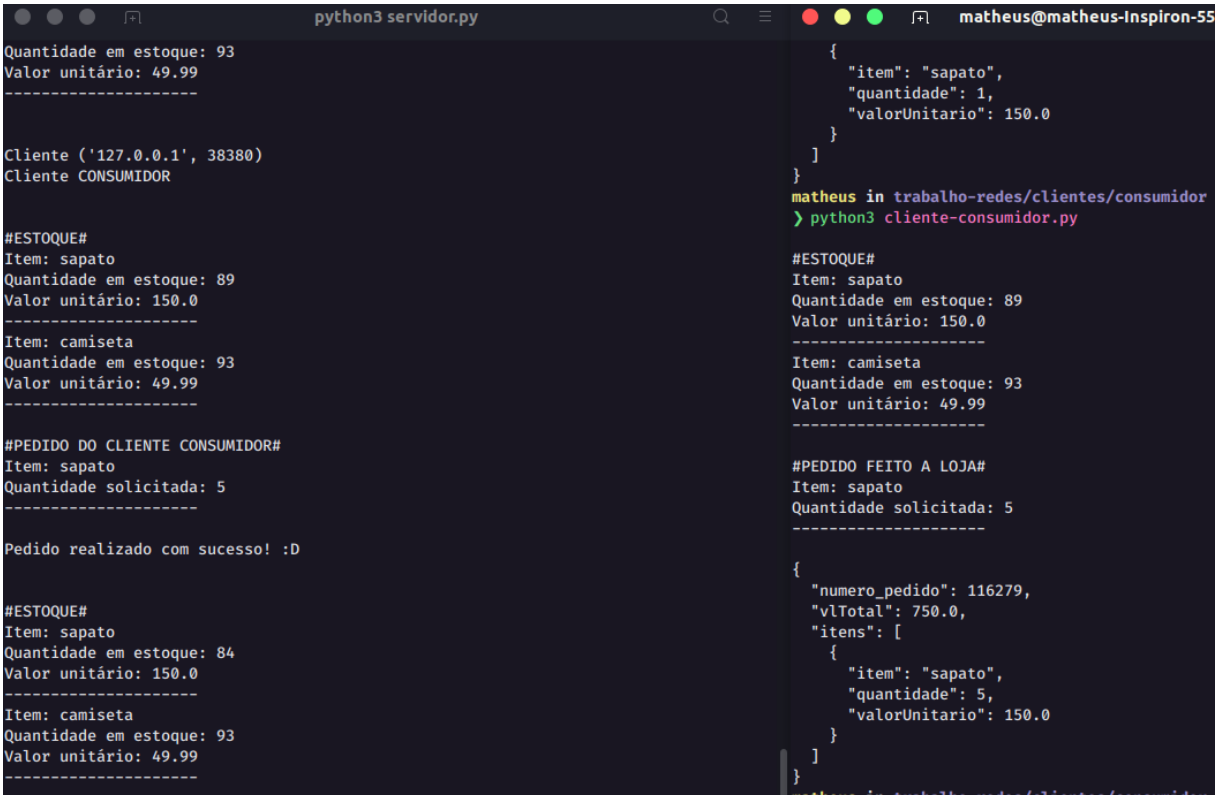
Serra
2021

1. Testes

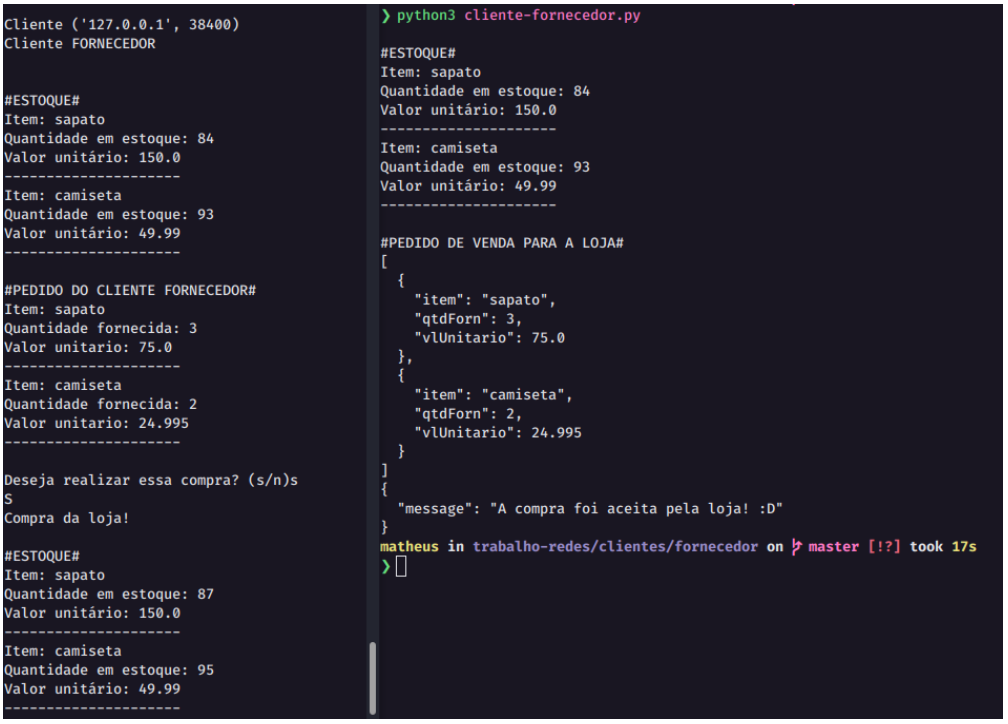
Teste 1 - Primeira compra do consumidor



Teste 2 - Segunda interação de compra



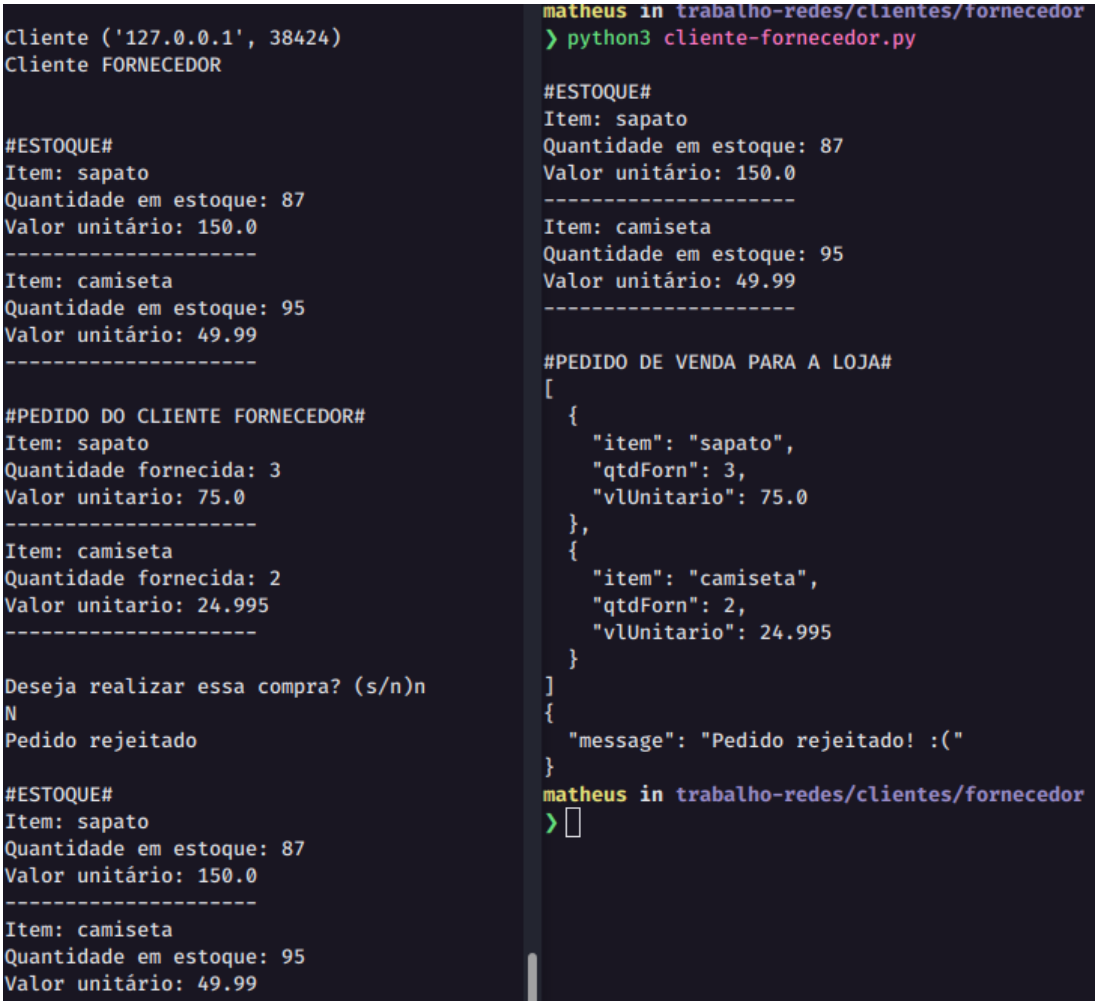
Teste 3 - Compra aceita pela loja



```
Cliente ('127.0.0.1', 38400)
Cliente FORNECEDOR


#ESTOQUE#
Item: sapato
Quantidade em estoque: 84
Valor unitário: 150.0
--------------------
Item: camiseta
Quantidade em estoque: 93
Valor unitário: 49.99
--------------------

#PEDIDO DO CLIENTE FORNECEDOR#
Item: sapato
Quantidade fornecida: 3
Valor unitario: 75.0
--------------------
Item: camiseta
Quantidade fornecida: 2
Valor unitario: 24.995
--------------------

Deseja realizar essa compra? (s/n)s
S
Compra da loja!

#ESTOQUE#
Item: sapato
Quantidade em estoque: 87
Valor unitário: 150.0
--------------------
Item: camiseta
Quantidade em estoque: 95
Valor unitário: 49.99
--------------------
```

```
❯ python3 cliente-fornecedor.py

#ESTOQUE#
Item: sapato
Quantidade em estoque: 84
Valor unitário: 150.0
--------------------
Item: camiseta
Quantidade em estoque: 93
Valor unitário: 49.99
--------------------

#PEDIDO DE VENDA PARA A LOJA#
[
  {
    "item": "sapato",
    "qtdForn": 3,
    "vlUnitario": 75.0
  },
  {
    "item": "camiseta",
    "qtdForn": 2,
    "vlUnitario": 24.995
  }
]
{
  "message": "A compra foi aceita pela loja! :D"
}
matheus in trabalho-redes/clientes/fornecedor on  master [!?] took 17s
❯
```

Teste 4 - Compra rejeitada pela loja



```
Cliente ('127.0.0.1', 38424)
Cliente FORNECEDOR


#ESTOQUE#
Item: sapato
Quantidade em estoque: 87
Valor unitário: 150.0
--------------------
Item: camiseta
Quantidade em estoque: 95
Valor unitário: 49.99
--------------------

#PEDIDO DO CLIENTE FORNECEDOR#
Item: sapato
Quantidade fornecida: 3
Valor unitario: 75.0
--------------------
Item: camiseta
Quantidade fornecida: 2
Valor unitario: 24.995
--------------------

Deseja realizar essa compra? (s/n)n
N
Pedido rejeitado

#ESTOQUE#
Item: sapato
Quantidade em estoque: 87
Valor unitário: 150.0
--------------------
Item: camiseta
Quantidade em estoque: 95
Valor unitário: 49.99
```

```
matheus in trabalho-redes/clientes/fornecedor
❯ python3 cliente-fornecedor.py

#ESTOQUE#
Item: sapato
Quantidade em estoque: 87
Valor unitário: 150.0
--------------------
Item: camiseta
Quantidade em estoque: 95
Valor unitário: 49.99
--------------------

#PEDIDO DE VENDA PARA A LOJA#
[
  {
    "item": "sapato",
    "qtdForn": 3,
    "vlUnitario": 75.0
  },
  {
    "item": "camiseta",
    "qtdForn": 2,
    "vlUnitario": 24.995
  }
]
{
  "message": "Pedido rejeitado! :("
}
matheus in trabalho-redes/clientes/fornecedor
❯
```

2. Instruções de execução

1- Clonar repositório https://github.com/matheuss3/trabalho-redes.git ou extrair arquivo .zip enviado
2- Para executar o servidor, abrir terminal na pasta trabalho-redes/servidor e executar o comando python3 servidor.py
3- Para executar o cliente consumidor abrir terminal na pasta trabalho-redes/clientes/consumidor e executar o comando python3 cliente-consumidor.py
4- Para executar o cliente consumidor abrir terminal na pasta trabalho-redes/clientes/fornecedor e executar o comando python3 cliente-fornecedor.py

3. Conclusão

Com este trabalho foi possível entender na prática os assuntos estudados em aula, utilizando a api de socket do python construímos um protocolo (utilizando uma conexão do tipo TCP) específico para venda, compra e gerência de estoque dos produtos de uma loja de conteúdo esportivos, através desse protocolo toda comunicação é feita entre a loja e seus consumidores e fornecedores.
Portanto fica clara a necessidade de possuir regras bem estabelecidas pelos protocolos, pois é com eles que as comunicações entre as redes são feitas. É possível perceber também a dificuldade de fazer essa comunicação de uma forma eficaz, pois para um problema existem n soluções nessa camada onde estávamos programando, nós éramos os responsáveis por estabelecer as regras, deste modo uma liberdade maior foi obtida na hora de escrever o código, mas, em contrapartida é necessário muito tempo pra pensar em como os dados serão transmitidos, qual formato, se precisarão ser serializados, compactados, codificados etc. Além desse problema é necessário ficar muito atento ao codificar, porque ao criar um *send* ou *recieve* no servidor se faz necessário receber ou enviar dados através dos clientes.
Para a transmissão dos dados utilizei o padrão JSON e a sua própria biblioteca em python para serializar e enviar os dados através dos clientes e do servidor, sendo assim quando alterava qualquer atributo ou tipo de favor na minha estrutura praticamente todo meu código apresentava problemas e todo ele precisaria ser ajustado mostrando um forte acoplamento entre as entidades o que é ruim para uma manutenção futura.
Apesar dos problemas, o algoritmo funcionou de forma satisfatória de acordo com as figuras apresentadas anteriormente neste documento, cumprindo com seu objetivo de gerenciar o estoque da loja e receber as requisições dos clientes.