

# Sistema Bancário Distribuído

**Discentes:** Héron Rezende e Kennedy Andrade

**Docente:** Raimundo José Macêdo

**Semestre:** 2023.2

# Sumário

01

## Servidor Bancário

Implementação do server.py

...

02

## Caixa Eletrônico

Implementação do client.py

...

03

## Utilidades Compartilhadas

O que há no utils.py

...

04

## Dúvidas e Perguntas

Momento de Perguntas

...

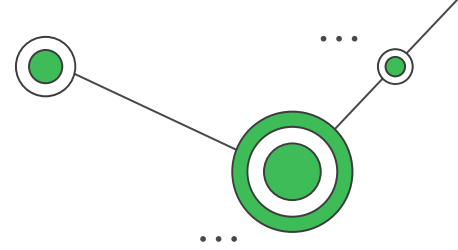


# 01

# Servidor

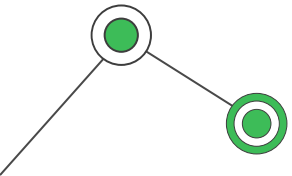
*server.py*

# Server.py



```
You, 1 second ago | 1 author (You)
1  import os
2  import json
3  import socket
4  import threading
5  from _thread import start_new_thread
6  from json import JSONDecodeError
7  from utils import OperacaoBancaria
8  from utils import OrigemRequisicao
9  from utils import StatusRequisicao
10 from utils import HOST
11 from utils import PORT
12 from utils import CONTAS_CORRENTES_DEFAULT
13 from utils import MAX_CLIENTS_CONNECTED
14 from utils import printar_valor_relogio_logico
15
```

```
9  # Configurações Iniciais
10 time = 0
11 contas_correntes = {}
12 mutex_time = threading.Lock()
13 mutex_accounts = threading.Lock()
14 server_socket = socket.socket()
15 thread_count = 0
16 threads = []
```



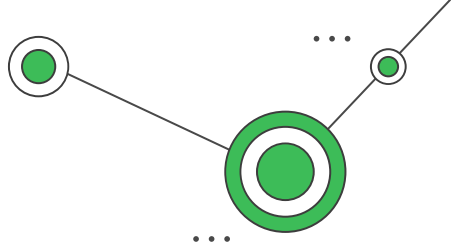
# Server.py

```
18  ▾ def incrementar_relogio():
19      global time
20      mutex_time.acquire()
21      time += 1
22      printar_valor_relogio_logico(OrigemRequisicao.SERVIDOR_BANCO.value, time)
23      mutex_time.release()
24
25  ▾ def ajustar_relogio_timestamp(timestamp, atual):
26      global time
27      mutex_time.acquire()
28      time = max(timestamp, atual) + 1
29      printar_valor_relogio_logico(OrigemRequisicao.SERVIDOR_BANCO.value, time)
30      mutex_time.release()
```

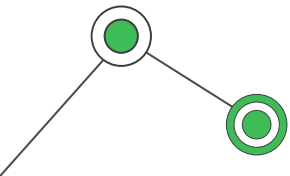
# Server.py

```
32 ✓ def salvar_contas(contas_correntes):
33 ✓     with open("contas_correntes.json", "w") as output_file:
34         json.dump(contas_correntes, output_file)
35
36 ✓ def carregar_contas():
37 ✓     try:
38         global contas_correntes
39 ✓         with open("contas_correntes.json", "r") as input_file:
40             contas_correntes = json.load(input_file)
41 ✓     except (FileNotFoundError):
42 ✓         print("{} - Não foi possível recuperar as contas correntes do arquivo."
43             .format(OrigemRequisicao.SERVIDOR_BANCO.value))
44 ✓         print("{} - Utilizaremos as contas default que estão disponíveis no arquivo 'utils.py'."
45             .format(OrigemRequisicao.SERVIDOR_BANCO.value))
46         contas_correntes = CONTAS_CORRENTES_DEFAULT
47         salvar_contas(contas_correntes)
48     return contas_correntes
```

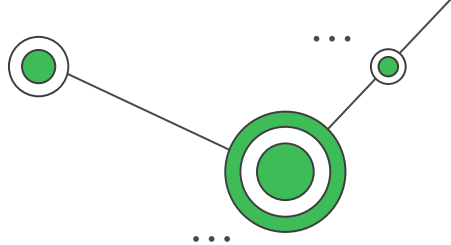
# Server.py



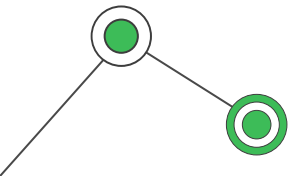
```
66 def realizar_saque(id_conta, valor_a_sacar):
67     if (contas_correntes[id_conta]['saldo'] - float(valor_a_sacar) >= 0):
68         contas_correntes[id_conta]['saldo'] -= float(valor_a_sacar)
69         salvar_contas(contas_correntes)
70         return "{} - Saque realizado!".format(OrigemRequisicao.SERVIDOR_BANCO.value)
71     else:
72         return "{} - Não foi possível realizar o saque pois o saldo é insuficiente.".format(OrigemRequisicao.SERVIDOR_BANCO.value)
73
74 def realizar_transferencia(id_conta_origem, id_conta_destino, valor_a_transferir):
75     if (contas_correntes[id_conta_origem]['saldo'] - float(valor_a_transferir) >= 0):
76         if (id_conta_destino not in contas_correntes):
77             return OrigemRequisicao.SERVIDOR_BANCO.value + ' - Não foi possível realizar a transferência pois a conta de destino não existe.'
78         contas_correntes[id_conta_origem]['saldo'] -= float(valor_a_transferir)
79         contas_correntes[id_conta_destino]['saldo'] += float(valor_a_transferir)
80         salvar_contas(contas_correntes)
81         return OrigemRequisicao.SERVIDOR_BANCO.value + ' - Transferência realizada!'
82     else:
83         return OrigemRequisicao.SERVIDOR_BANCO.value + ' - Não foi possível realizar a transferência pois o saldo da conta de origem é insuficiente.'
```



# Server.py



```
66 def realizar_saque(id_conta, valor_a_sacar):
67     if (contas_correntes[id_conta]['saldo'] - float(valor_a_sacar) >= 0):
68         contas_correntes[id_conta]['saldo'] -= float(valor_a_sacar)
69         salvar_contas(contas_correntes)
70         return "{} - Saque realizado!".format(OrigemRequisicao.SERVIDOR_BANCO.value)
71     else:
72         return "{} - Não foi possível realizar o saque pois o saldo é insuficiente.".format(OrigemRequisicao.SERVIDOR_BANCO.value)
73
74 def realizar_transferencia(id_conta_origem, id_conta_destino, valor_a_transferir):
75     if (contas_correntes[id_conta_origem]['saldo'] - float(valor_a_transferir) >= 0):
76         if (id_conta_destino not in contas_correntes):
77             return OrigemRequisicao.SERVIDOR_BANCO.value + ' - Não foi possível realizar a transferência pois a conta de destino não existe.'
78         contas_correntes[id_conta_origem]['saldo'] -= float(valor_a_transferir)
79         contas_correntes[id_conta_destino]['saldo'] += float(valor_a_transferir)
80         salvar_contas(contas_correntes)
81         return OrigemRequisicao.SERVIDOR_BANCO.value + ' - Transferência realizada!'
82     else:
83         return OrigemRequisicao.SERVIDOR_BANCO.value + ' - Não foi possível realizar a transferência pois o saldo da conta de origem é insuficiente.'
```

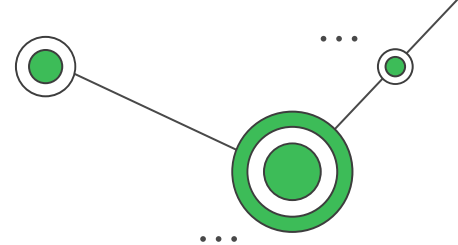




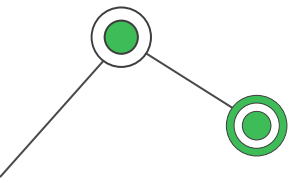
# Server.py

```
174 def main():
175     inicializar_servidor()
176
177     while True:
178         try:
179             client, address = server_socket.accept()
180             print("{} - Conectado ao cliente {}:{}".format(OrigemRequisicao.SERVIDOR_BANCO.value, address[0], str(address[1])))
181             threads.append(start_new_thread(threaded_client, (client, )))
182             global thread_count
183             thread_count += 1
184             print("{} - Thread número: {}".format(OrigemRequisicao.SERVIDOR_BANCO.value, str(thread_count)))
185         except Exception as exc:
186             print(exc)
187             break
188
189     server_socket.close()
190
191 if __name__ == "__main__":
192     main()
193
```

# Server.py



```
161  def inicializar_servidor():
162      try:
163          carregar_contas()
164          server_socket.bind((HOST, PORT))
165          print("{} - Aguardando conexão de clients... (MAX = {})".format(OrigemRequisicao.SERVIDOR_BANCO.value, MAX_CLIENTS_CONNECTED))
166          server_socket.listen(MAX_CLIENTS_CONNECTED)
167          global time
168          time += 1
169          printar_valor_relogio_logico(OrigemRequisicao.SERVIDOR_BANCO.value, time)
170      except socket.error as err:
171          print("{} - Não foi possível inicializar o servidor!", OrigemRequisicao.SERVIDOR_BANCO.value)
172          print(str(err))
173
```



# Server.py

```
86 def threaded_client(connection):  
87     # Confirma a conexão realizada com o client.  
88     global time  
89     global contas_correntes  
90     incrementar_relogio()  
91     enviar_mensagem(connection, {'message': 'Conexão estabelecida com sucesso!', 'time': time, 'status': StatusRequisicao.OK.value})  
92  
93     # Recebe do client o identificador da conta corrente.  
94     response = receber_resposta(connection)  
95     if response['status'] == 500:  
96         connection.close()  
97         return  
98  
99     id_conta = response['identificador_origem']  
100     ajustar_relogio_timestamp(int(response['time']), time)  
101     incrementar_relogio()  
102  
103     # Verifica se a chave da conta corrente recebida do client existe no dicionário.  
104     if id_conta in contas_correntes:  
105         text_message = OrigemRequisicao.SERVIDOR_BANCO.value + ' - Conta corrente encontrada.'  
106         enviar_mensagem(connection, {'message': text_message, 'time': time, 'status': StatusRequisicao.OK.value})  
107     else:  
108         text_message = OrigemRequisicao.SERVIDOR_BANCO.value + ' - Esta conta corrente não foi encontrada.'  
109         enviar_mensagem(connection, {'message': text_message, 'time': time, 'status': StatusRequisicao.NOT_FOUND.value})  
110     return
```



# Server.py

```
while True:
    response = receber_resposta(connection)
    if response['status'] == 500:
        break

    if response['operation'] == OperacaoBancaria.SALDO.value:
        ajustar_relogio_timestamp(int(response['time']), time)
        carregar_contas()
        incrementar_relogio()
        incrementar_relogio()
        text_message = OrigemRequisicao.SERVIDOR_BANCO.value + ' - Saldo disponível: R$' + str(contas_correntes[id_conta]['saldo'])
        enviar_mensagem(connection, {'message': text_message, 'time': time, 'status': StatusRequisicao.OK.value})
    elif response['operation'] == OperacaoBancaria.DEPOSITO.value:
        ajustar_relogio_timestamp(int(response['time']), time)
        mutex_accounts.acquire()
        carregar_contas()
        contas_correntes[id_conta]['saldo'] += float(response['value'])
        salvar_contas(contas_correntes)
        mutex_accounts.release()
        incrementar_relogio()
        incrementar_relogio()
        text_message = OrigemRequisicao.SERVIDOR_BANCO.value + ' - Depósito realizado!'
        enviar_mensagem(connection, {'message': text_message, 'time': time, 'status': StatusRequisicao.CREATED.value})
```

# Server.py


```
135 elif response['operation'] == OperacaoBancaria.SAQUE.value:
136     ajustar_relogio_timestamp(int(response['time']),time)
137     mutex_accounts.acquire()
138     carregar_contas()
139     saque_response_message = realizar_saque(id_conta, response['value'])
140     mutex_accounts.release()
141     incrementar_relogio()
142     incrementar_relogio()
143     enviar_mensagem(connection, {'message': saque_response_message, 'time': time, 'status': StatusRequisicao.CREATED.value})
144 elif response['operation'] == OperacaoBancaria.TRANSFERENCIA.value:
145     ajustar_relogio_timestamp(int(response['time']),time)
146     mutex_accounts.acquire()
147     carregar_contas()
148     transferencia_response_message = realizar_transferencia(id_conta, response['identificador_destino'], response['value'])
149     mutex_accounts.release()
150     incrementar_relogio()
151     incrementar_relogio()
152     enviar_mensagem(connection, {'message': transferencia_response_message, 'time': time, 'status': StatusRequisicao.CREATED.value})
153 elif response['operation'] == OperacaoBancaria.DESCONECTAR.value:
154     ajustar_relogio_timestamp(int(response['time']),time)
155     incrementar_relogio()
156     text_message = OrigemRequisicao.SERVIDOR_BANCO.value + ' - Encerrando conexão...'
157     enviar_mensagem(connection, {'message': text_message, 'time': time, 'status': StatusRequisicao.OK.value})
158     break
159 connection.close()
```



02

# Caixa Eletrônico

*client.py*

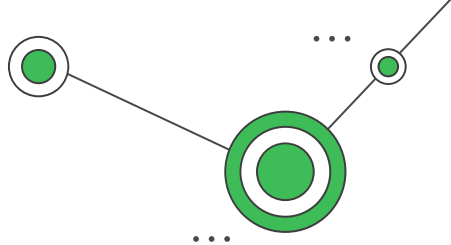


# *Client.py*

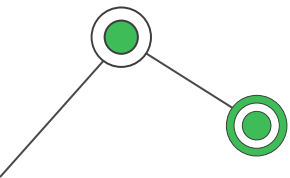
```
1  import socket
2  import json
3  import sys
4  \ from utils import OperacaoBancaria
5  \ from utils import OrigemRequisicao
6  \ from utils import StatusRequisicao
7  \ from utils import HOST
8  \ from utils import PORT
9  \ from utils import printar_valor_relogio_logico
10
11  # Configurações iniciais do cliente
12  client_socket = socket.socket()
13  time = 0
```



# Client.py

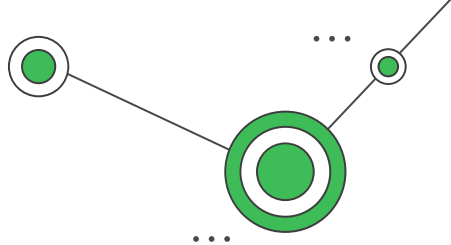


```
15  ✓ def ajustar_valor_relogio_logico(server_time):
16      """Função para ajustar o valor do relógio lógico com base no timer do server."""
17      global time
18      time = max(time, int(server_time)) + 1
19      printar_valor_relogio_logico(OrigemRequisicao.CAIXA_ELETRONICO.value, time)
20
21  ✓ def incrementar_valor_relogio_logico():
22      """Função para incrementar o valor do relógio lógico."""
23      global time
24      time += 1
25      printar_valor_relogio_logico(OrigemRequisicao.CAIXA_ELETRONICO.value, time)
```

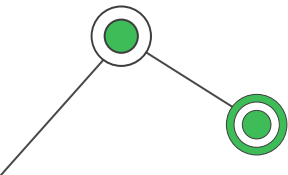




# Client.py



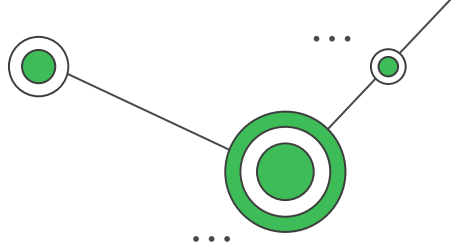
```
27 def estabelecer_conexao():
28     """Estabelece a conexão com o servidor."""
29     try:
30         client_socket.connect((HOST, PORT))
31         resposta_conexao = receber_resposta()
32         print("{} - {}".format(OrigemRequisicao.SERVIDOR_BANCO.value, resposta_conexao['message']))
33     except socket.error as err:
34         print("{} - Não foi possível estabelecer a conexão com o banco... Tente novamente mais tarde!"
35               .format(OrigemRequisicao.CAIXA_ELETRONICO.value))
36         print(str(err))
37         sys.exit(1)
```



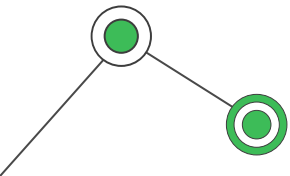
# *Client.py*

```
39  def enviar_mensagem(data):
40      """Envia uma mensagem ao servidor."""
41      incrementar_valor_relogio_logico()
42      body = json.dumps(data)
43      client_socket.send(body.encode('utf-8'))
44
45  def receber_resposta():
46      """Recebe a resposta do servidor."""
47      response = client_socket.recv(1024).decode('utf-8')
48      body = json.loads(response)
49      ajustar_valor_relogio_logico(body['time'])
50      return body
51
```

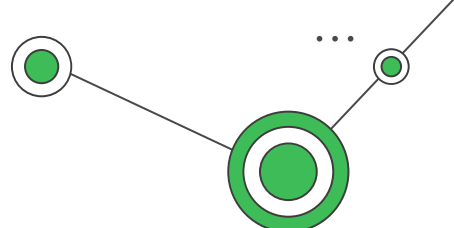
# Client.py



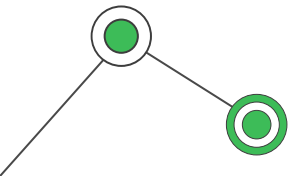
```
47  def main():
48      # Realiza conexão inicial com o servidor
49      estabelecer_conexao()
50
51      # Enviar RG/CPF ao servidor
52      try:
53          identificador_pessoa = input('[Caixa Eletrônico] - Digite seu CPF ou RG: ')
54          enviar_mensagem({'identificador_origem': identificador_pessoa, 'time': time, 'status': StatusRequisicao.OK.value})
55          resposta_identificador = receber_resposta()
56          print(resposta_identificador['message'])
57          if(resposta_identificador['status'] == 404):
58              print("{} - Por favor, tente novamente mais tarde.".format(OrigemRequisicao.CAIXA_ELETRONICO.value))
59              sys.exit(1)
60      except (KeyboardInterrupt, ConnectionResetError):
61          print("\n{} - Operação cancelada.".format(OrigemRequisicao.CAIXA_ELETRONICO.value))
62          sys.exit(1)
```



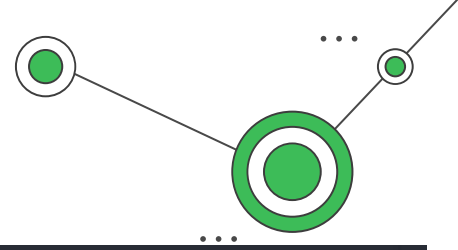
# Client.py



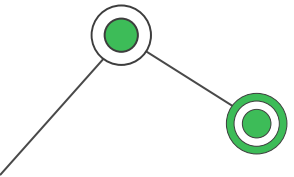
```
64 while True:
65     try:
66         print("{} - Escolha uma das operações abaixo:".format(OrigemRequisicao.CAIXA_ELETRONICO.value))
67         for operacao in OperacaoBancaria:
68             print(f"{operacao.value}. {operacao.name.capitalize()}")
69         op = input()
70
71         if op == OperacaoBancaria.SALDO.value:
72             print('{} - Consultando saldo...'.format(OrigemRequisicao.CAIXA_ELETRONICO.value))
73             enviar_mensagem({'time': time, 'operation': op, 'status': StatusRequisicao.OK.value})
74             resposta_saldo = receber_resposta()
75             print(resposta_saldo['message'])
76         elif op == OperacaoBancaria.SAQUE.value:
77             saque = input('[Caixa Eletrônico] - Digite o valor que deseja sacar: ')
78             enviar_mensagem({'time': time, 'operation': op, 'value': saque, 'status': StatusRequisicao.OK.value})
79             resposta_saque = receber_resposta()
80             print(resposta_saque['message'])
```



# Client.py



```
81  elif op == OperacaoBancaria.DEPOSITO.value:
82      deposito = input('[Caixa Eletrônico] - Digite o valor que deseja depositar: ')
83      enviar_mensagem({'time': time, 'operation': op, 'value': deposito, 'status': StatusRequisicao.OK.value})
84      resposta_deposito = receber_resposta()
85      print(resposta_deposito['message'])
86  elif op == OperacaoBancaria.TRANSFERENCIA.value:
87      identificador_transferencia = input('[Caixa Eletrônico] - Digite o identificador da conta para a qual quer transferir: ')
88      value_transferencia = input('[Caixa Eletrônico] - Digite o valor que deseja transferir: ')
89      enviar_mensagem({'identificador_destino': identificador_transferencia, 'time': time, 'operation': op, 'value': value_transferencia, 'status': StatusR
90      resposta_transferencia = receber_resposta()
91      print(resposta_transferencia['message'])
92  elif op == OperacaoBancaria.DESCONNECTAR.value:
93      enviar_mensagem({'time': time, 'operation': op, 'status': StatusRequisicao.OK.value})
94      print("{} - Obrigado por utilizar os nossos serviços!".format(OrigemRequisicao.CAIXA_ELETRONICO.value))
95      break
96  else:
97      print("{} - Comando inválido, tente novamente!".format(OrigemRequisicao.CAIXA_ELETRONICO.value))
98  except (KeyboardInterrupt, ConnectionResetError):
99      break
100
101  client_socket.close()
```



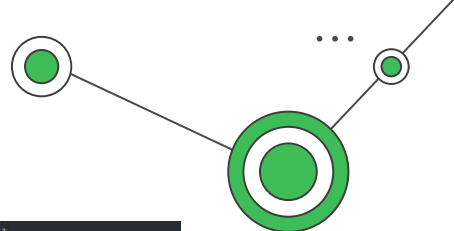
# 03

## Utilidades

*utils.py*



# Utils.py




```
3  # Definindo constantes de utilidades e configurações
4  HOST = '127.0.0.1'
5  PORT = 7667
6  MAX_CLIENTS_CONNECTED = 5
7  CONTAS_CORRENTES_DEFAULT = {
8      '123': {'nome': 'Kennedy Anderson', 'saldo': 0.0},
9      '456': {'nome': 'Herson Rezende', 'saldo': 0.0},
10     '789': {'nome': 'Raimundo Macedo', 'saldo': 0.0}
11 }
```

```
13 # Definindo uma enumeração chamada OperacaoBancaria
    You, 3 weeks ago | 1 author (You)
14 class OperacaoBancaria(Enum):
15     SALDO = '0'
16     DEPOSITO = '1'
17     SAQUE = '2'
18     TRANSFERENCIA = '3'
19     DESCONECTAR = '99'
```

```
    You, 3 weeks ago | 1 author (You)
26 class StatusRequisicao(Enum):
27     OK = 200
28     CREATED = 201
29     BAD_REQUEST = 400
30     NOT_FOUND = 404
31     INTERNAL_SERVER_ERROR = 500
```

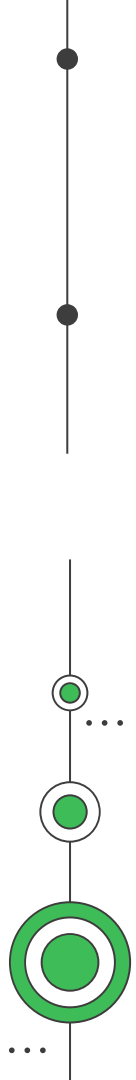
```
21 # Definindo uma enumeração chamada OrigemRequisicao
    You, 3 weeks ago | 1 author (You)
22 class OrigemRequisicao(Enum):
23     CAIXA_ELETRONICO = '[Caixa Eletrônico]'
24     SERVIDOR_BANCO = '[Servidor Bancário]'
25
```

```
33 # Funções auxiliares
34 def printar_valor_relogio_logico(origem, tempo):
35     """Função para imprimir o valor do relógio lógico."""
36     mensagem = "{} - Valor atual do relógio lógico: {}."
37     print(mensagem.format(origem, tempo))
```



04

Perguntas







**Obrigado!**

