

Mecanismos de Comunicação em Sistemas Distribuídos

Prof. Marcos Momo
Prof. Ricardo J. Rabelo

abril/2024

Trabalho 1 – Exercício de Implementação

- 15% da Nota na disciplina
- Trabalho individual
- Entrega até 7/5 às 22:00
- Fazer um zip (ou criar link) com todos os códigos
- Enviar por email para Marcos Rodrigo Momo (marcos.rodrigomomo@gmail.com)

Esse exercício dá bem importantes bases de conhecimento para o posterior desenvolvimento do trabalho final da disciplina.

Sumário

- Socket
- Websocket
- Banco de dados
- Invocação Remota de Procedimentos
- File Transfer
- Rest API AWS Lambda

Instalação Python e pip Windows:

1. Baixar instalador: <https://www.python.org>

Versão Python 3.12.3

2. Executar o instalador baixado

Marcar a opção pip no momento da instalação

Marcar a opção Add Python to enviroment variables no momento da instalação

3. Executar no Powershell para validar a instalação: `python - -version`

4. Executar no Powershell para validar a instalação: `pip - -version`

Instalação da IDE e dependências

Instalação VSCode: <https://code.visualstudio.com/download>

Instalação Postman: <https://dl.pstmn.io/download/latest/win64>

Instalação do websocket-client: <https://pypi.org/project/websocket-client/>

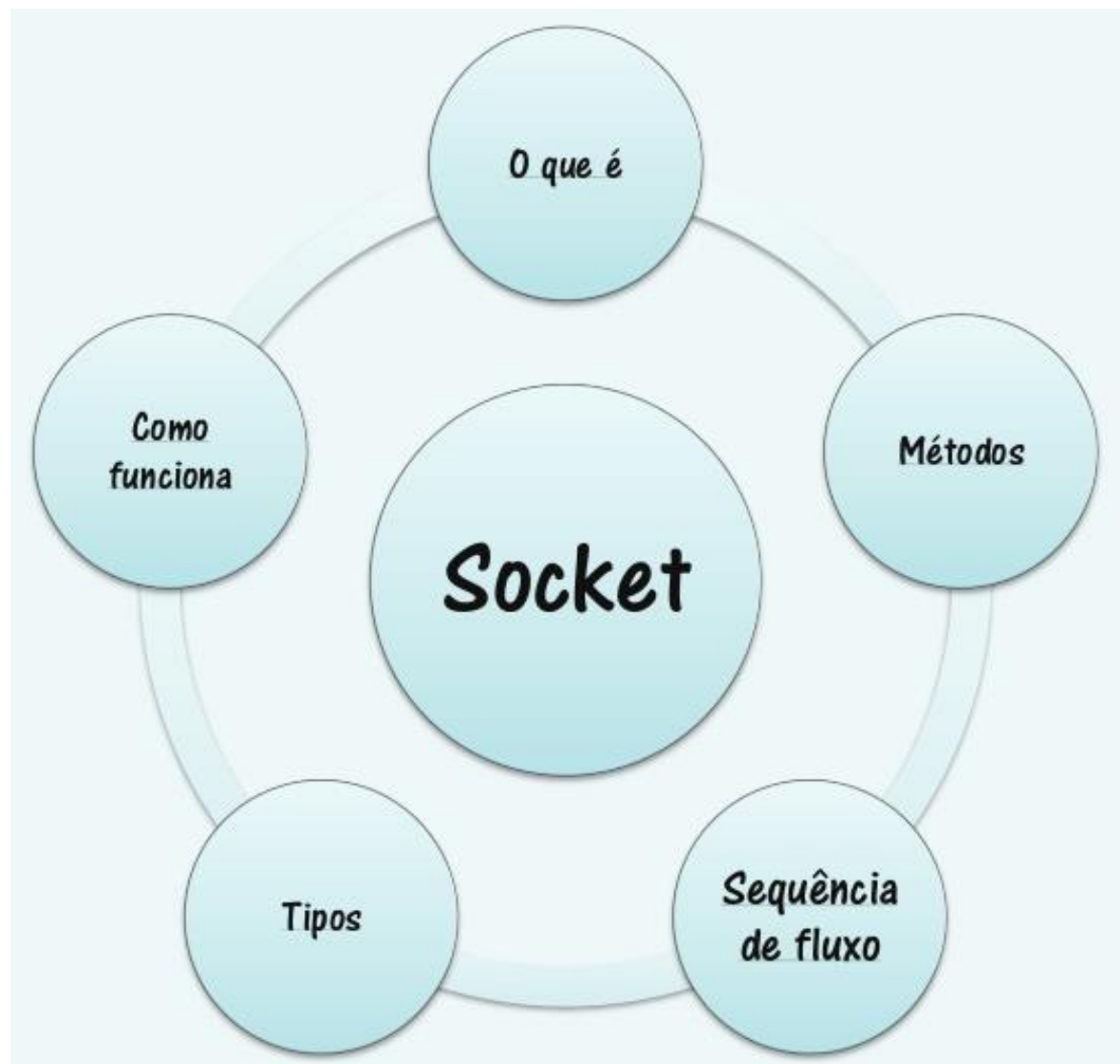
Instalação Requests: <https://pypi.org/project/requests/>

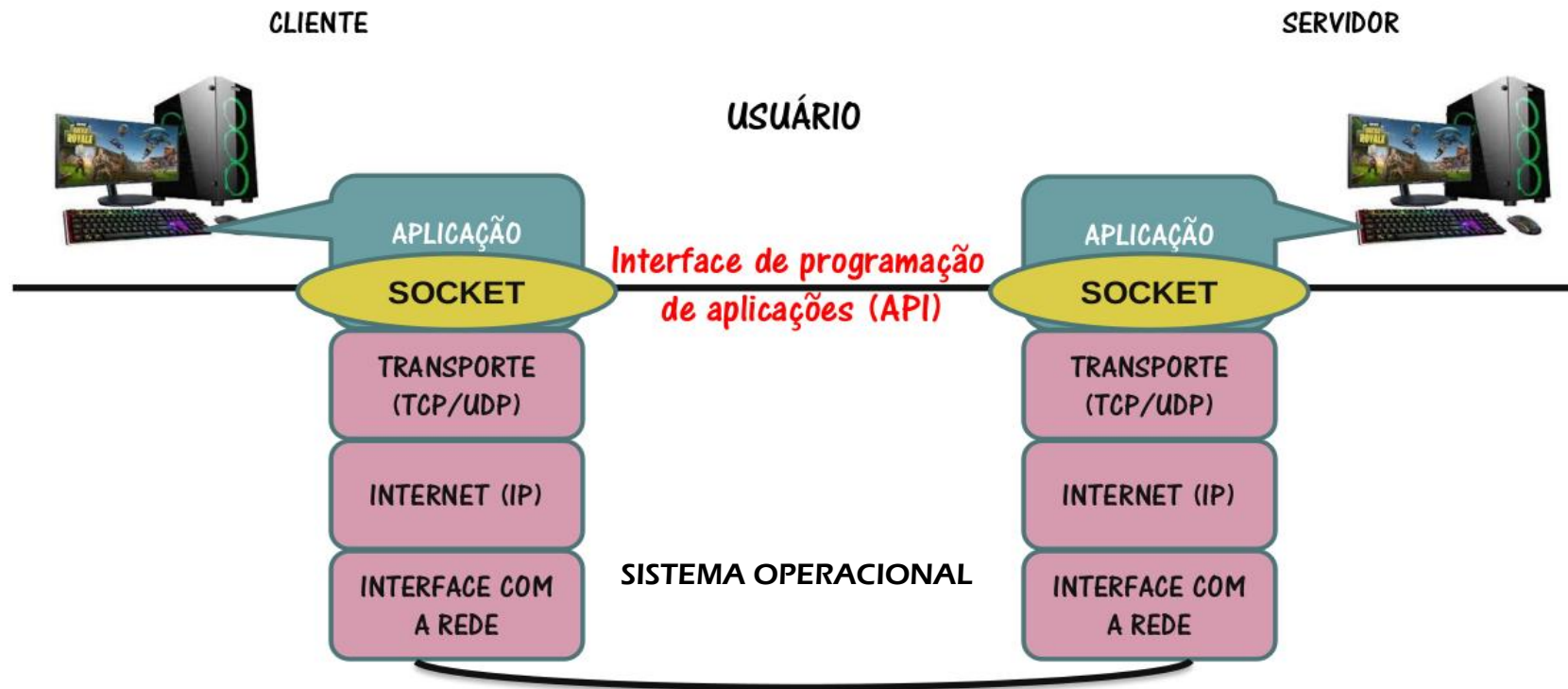
Instalação Chalice: <https://pypi.org/project/chalice/>

SQLiteStudio: <https://sqlitestudio.pl>

Sumário

- **Socket**
- Websocket
- Banco de dados
- Invocação Remota de Procedimentos
- File Transfer
- Rest API AWS Lambda





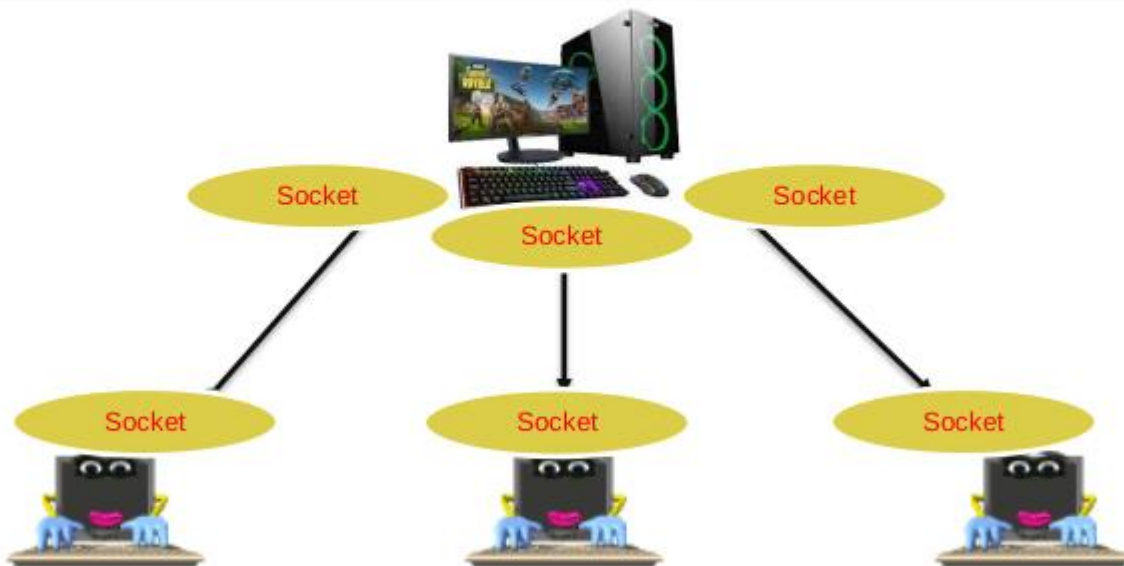
O que é

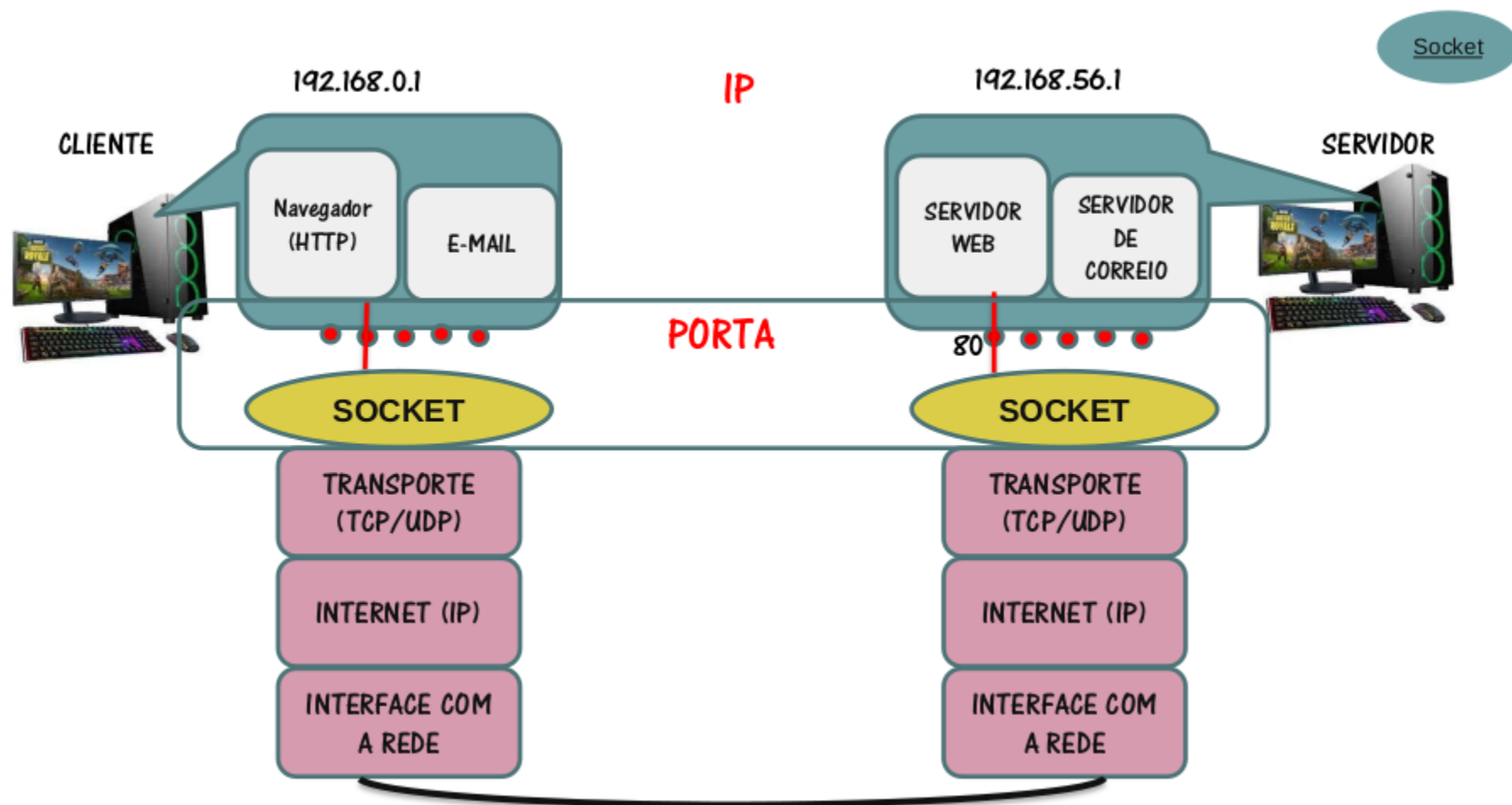
São **interfaces programáveis** de comunicação entre softwares que podem estar rodando em computadores distintos na rede. Eles permitem transferir strings em bytes de um processo para outro e é a base da maioria dos protocolos de alto nível, como FTP, páginas da Web, e e-mail.



O que é

Imagine por exemplo que precisa desenvolver uma aplicação servidor que funcione como uma “sala” de chat. Na prática essa aplicação irá receber ligações dos clientes e, posteriormente, se um cliente enviar uma mensagem, o servidor envia essa mensagem para todos os outros clientes. É necessário um protocolo capaz de enviar e receber as mensagens.

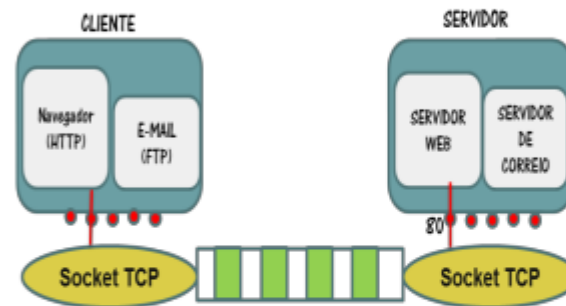




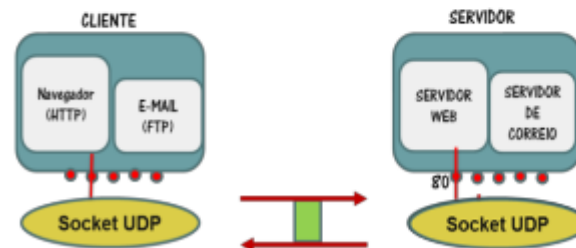
- `socket()`
- `bind()`
- `listen()`
- `accept()`
- `connect()`
- `connect_ex()`
- `send()`
- `recv()`
- `close()`

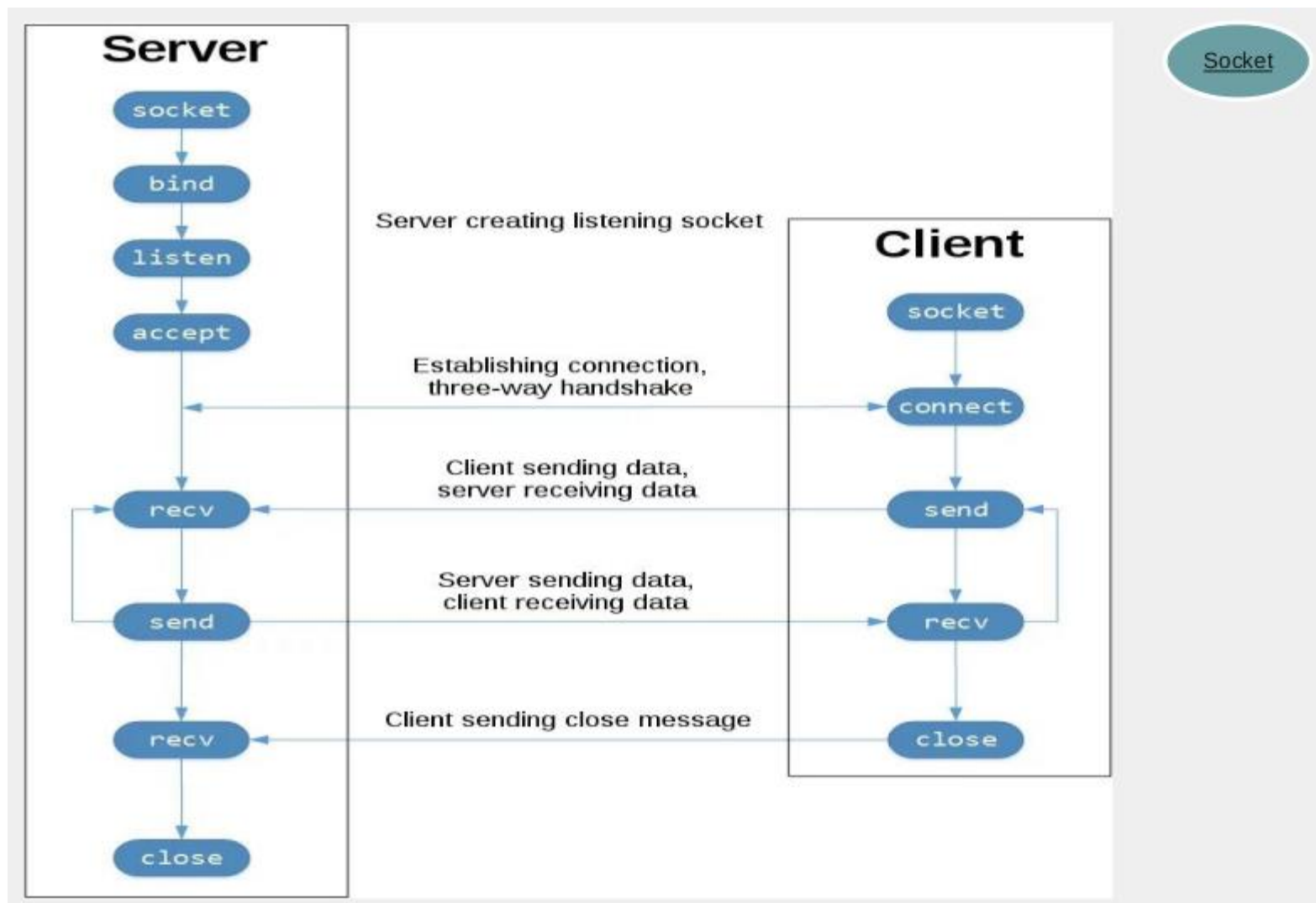
Para a comunicação na rede, o Socket opera por **protocolo de comunicação TCP** ou UDP para abstrair as complexidades inerentes do meio de transmissão.

- TCP:**
- Orientado à conexão, comunicação fiável e ordenada;
 - As mensagens são vistas pelo nível de transporte como fluxo de dados;
 - As mensagens são de tipo STREAM.



- UDP:**
- Não tem garantia de entrega;
 - Orientado para mensagens independentes;
 - As mensagens são de tipo DGRAM.





Socket - Exemplo de implementação

Server.py

```
1  from socket import *
2
3  meuHost = '127.0.0.1'
4  minhaPort = 50009
5
6  sockobj = socket(AF_INET, SOCK_STREAM)
7  sockobj.bind((meuHost, minhaPort))
8  sockobj.listen(1)
9
10 while True:
11     conexao, endereco = sockobj.accept()
12     print('Server conectado por', endereco)
13
14     while True:
15         data = conexao.recv(1024)
16         print('Cliente enviou:', data.decode())
17         resposta = 'Eco=>' + data.decode()
18         conexao.send(resposta.encode())
19
20 conexao.close()
```

Client.py

```
1  from socket import *
2
3  serverHost = '127.0.0.1'
4  serverPort = 50009
5
6  sockobj = socket(AF_INET, SOCK_STREAM)
7  sockobj.connect((serverHost, serverPort))
8
9  while True:
10     print('Digite uma mensagem')
11     msg = input()
12     sockobj.send(msg.encode())
13     data = sockobj.recv(1024)
14     print('Server recebeu:', data.decode())
15
16 sockobj.close()
```

Socket - Exemplo de implementação

Exemplo saída de execução

Saída no servidor

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

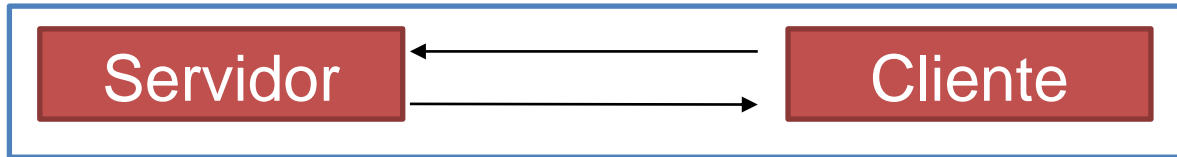
```
momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/scket$ python server.py
Server conectado por ('127.0.0.1', 54364)
Cliente enviou: Olá
Cliente enviou: Bom dia
Cliente enviou: Boa tarde
█
```

Saída no cliente

```
momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/ocket$ python client.py
Digite uma mensagem
Olá
Server recebeu: Eco=>Olá
Digite uma mensagem
Bom dia
Server recebeu: Eco=>Bom dia
Digite uma mensagem
Boa tarde
Server recebeu: Eco=>Boa tarde
Digite uma mensagem
█
```


Socket - Exercício de implementação

- Desenvolva, com o código fornecido, um chat no terminal entre o cliente e o servidor.
 - Requisito:
 - O cliente irá enviar a primeira mensagem.



Sumário

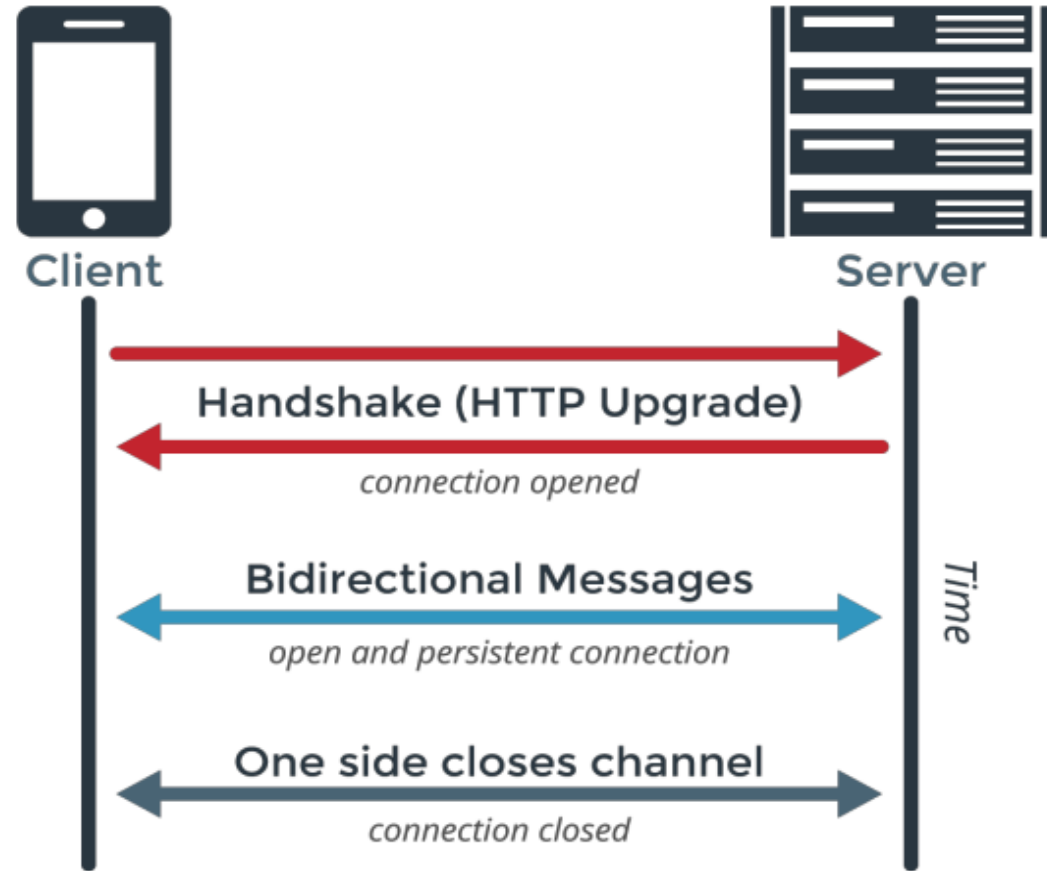
- Socket
- **Websocket**
- Banco de dados
- Invocação Remota de Procedimentos
- File Transfer
- Rest API AWS Lambda

WebSocket - O que é

- WebSockets são uma tecnologia avançada que torna possível a abertura de uma sessão de comunicação interativa entre o browser do usuário e servidor.
- Desta forma, é possível enviar mensagem para um servidor e receber respostas *event-driven* sem ter que esperar o servidor por uma resposta.

WebSocket:

O que é



WebSocket - Como funciona

O protocolo WebSocket possui a finalidade específica de ser implementado em navegadores web para comunicação com um servidor web, comunicando-se sobre o protocolo TCP.

O formato do protocolo é em texto e o *handshake* que inicia a comunicação é muito parecido com o HTTP, fazendo com que servidores web possam usar HTTP e WebSockets na mesma porta de comunicação.

WebSocket - Exemplo 1 de implementação

Server.py

```
1 import asyncio
2 import websockets
3
4 async def hello(websocket, path):
5     name = await websocket.recv()
6     print(f"< {name}")
7
8     greeting = f"Ola {name}!"
9
10    await websocket.send(greeting)
11    print(f"> {greeting}")
12
13    start_server = websockets.serve(hello, 'localhost', 8765)
14
15    asyncio.get_event_loop().run_until_complete(start_server)
16    asyncio.get_event_loop().run_forever()
```

Saída no servidor

```
TERMINAL  PROBLEMS 2  OUTPUT  DEBUG CONSOLE
momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/websocket$ python server.py
< Marcos
> Ola Marcos!
```

Client.py

```
1 import asyncio
2 import websockets
3
4 async def hello():
5
6     async with websockets.connect('ws://localhost:8765') as websocket:
7         name = input("Qual seu nome? ")
8
9         await websocket.send(name)
10        print(f"> {name}")
11
12        greeting = await websocket.recv()
13        print(f"< {greeting}")
14    asyncio.get_event_loop().run_until_complete(hello())
```

Entrada/Saída no cliente

```
SOLE python + v  ...  ^  x
momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/websocket$ python client.py
Qual seu nome? Marcos
> Marcos
< Ola Marcos!
momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/websocket$
```

WebSocket - Exemplo 2 de implementação

Server.py

```
1 import asyncio
2 import datetime
3 import random
4 import websockets
5 async def time(websocket, path):
6     while True:
7         now = input("Mensagem: ")
8         await websocket.send(now)
9         await asyncio.sleep(random.random() * 3)
10
11 start_server = websockets.serve(time, '127.0.0.1', 5678)
12 asyncio.get_event_loop().run_until_complete(start_server)
13 asyncio.get_event_loop().run_forever()
```

Saída no servidor

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/scket$ python server.py
Server conectado por ('127.0.0.1', 54364)
Cliente enviou: Olá
Cliente enviou: Bom dia
Cliente enviou: Boa tarde
█
```

Client.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>WebSocket demo</title>
5 </head>
6 <body>
7   <script>
8     var ws = new WebSocket("ws://127.0.0.1:5678/"),
9     messages = document.createElement('ul');
10    ws.onmessage = function (event) {
11      var messages = document.getElementsByTagName('ul')[0],
12      message = document.createElement('li'),
13      content = document.createTextNode(event.data);
14      message.appendChild(content);
15      messages.appendChild(message);
16    };
17    document.body.appendChild(messages);
18  </script>
19 </body>
20 </html>
```

Entrada/Saída no cliente

```
momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/ocket$ python client.py
```

```
5
Digite uma mensagem
Olá
Server recebeu: Eco=>Olá
Digite uma mensagem
Bom dia
Server recebeu: Eco=>Bom dia
Digite uma mensagem
Boa tarde
Server recebeu: Eco=>Boa tarde
Digite uma mensagem
█
```

WebSocket - Exercício de Implementação

A partir do código do exemplo 2, criar um servidor capaz de receber dois números, somar, e apresentar no browser a resposta.

Sumário

- Socket
- Websocket
- **Banco de dados**
- Invocação Remota de Procedimentos
- File Transfer
- Rest API AWS Lambda

Banco de Dados - O que é

- É a principal tecnologia para *persistência* (armazenamento físico) de (grandes volumes) de dados.
- Permitem organizar coleções de dados e realizar pesquisas sobre eles com grande eficiência.
- É a tecnologia usada em Cloud para armazenamento de dados.
- São operados pelos Sistemas Gerenciadores de Bancos de Dados (SGBD), que surgiram na década de 70. Antes destes, as aplicações usavam sistemas de arquivos do sistema operacional para armazenar suas informações. Na década de 80, a tecnologia de SGBD **relacional** passou a dominar o mercado, e atualmente é largamente usada.
- Outros tipos de BDs são os não-relacionais, XML, orientados a objetos, e os de tempo real. Os precursores / mais antigos BDs eram do tipo indexados.



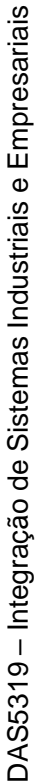
Banco de Dados - O que é

Um SGDB tem 4 componentes:

- API ou *driver*: interface de acesso ao BD
- Dicionário de Dados: esquema/templates das tabelas e campos que existem na BD
- Linguagem de Acesso: linguagem que permite um sistema externo [cliente] acessar o BD [sistema servidor] (e.g. SQL)
- Base de Dados: parte do SGDB onde estão os dados propriamente ditos



DAS5319 – Integração de Sistemas Industriais e Empresariais

DAS5319 – Integração de Sistemas Industriais e Empresariais

DAS5319 – Integração de Sistemas Industriais e Empresariais

Banco de Dados - O que é

[illegible]

1

[illegible]

2

[illegible]

3

[illegible]

4

[illegible]

5

Registros Acesso Direto

Banco de Dados Relacional – como funciona

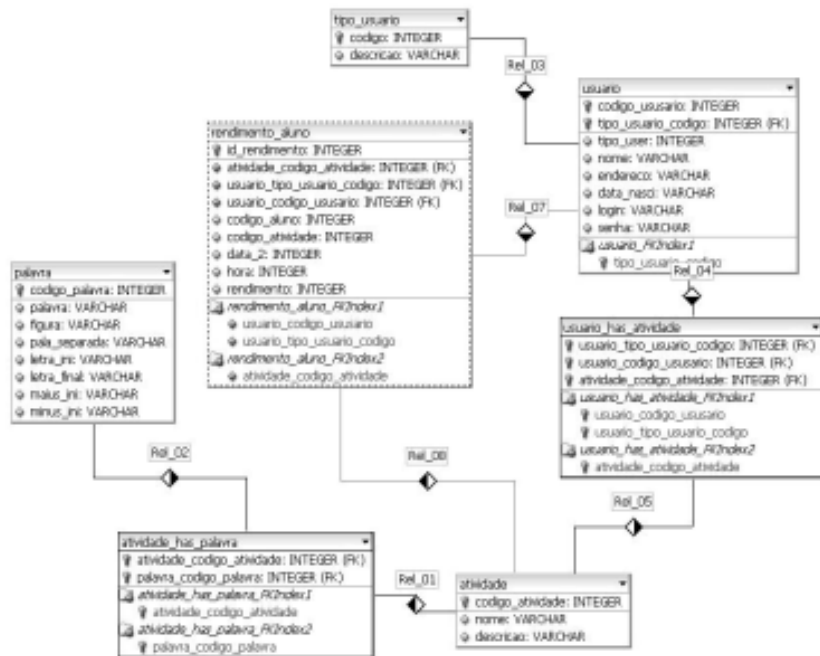


Table name: PEDIDOS		WITHOUT ROWID							
	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	
1	CODIGO	INTEGER	🔑						NULL
2	CLIENTE	VARCHAR (40)							NULL
3	ENDERECO	VARCHAR (100)							NULL
4	ESTADO	VARCHAR (10)							NULL
5	QTD_ROBO_DOMESTICO	INTEGER							NULL
6	QTD_REATOR_SOLAR	INTEGER							NULL
7	QTD_REATOR_ARK	INTEGER							NULL
8	TOTAL	REAL							NULL

Banco de Dados - Exemplo de implementação

Escrever.py

```
1 import sqlite3
2
3 conn = sqlite3.connect('aula.db')
4 c = conn.cursor()
5
6 def create_table():
7     c.execute(
8         "CREATE TABLE IF NOT EXISTS Alunos(Nome TEXT, Matricula REAL, Idade INTEGER, Ingresso INTEGER, Situacao TEXT)"
9     )
10
11 def data_entry():
12     c.execute("INSERT INTO Alunos VALUES('Pedro', 2213482, 19, 2022, 'Cursando')")
13
14 create_table()
15 data_entry()
16
17 conn.commit()
18 c.close()
19 conn.close()
```

Ler.py

```
1 import sqlite3
2
3 conn = sqlite3.connect('aula.db')
4 c = conn.cursor()
5
6 def read_from_db():
7     c.execute('SELECT * FROM Alunos')
8     data = c.fetchall()
9     for row in data:
10         print(row)
11
12     c.execute('SELECT Idade FROM Alunos WHERE Idade > 18')
13     data = c.fetchall()
14     for row in data:
15         print(row)
16
17 read_from_db()
18 c.close()
19 conn.close()
```

Saída da execução do arquivo Ler.py
após a execução do arquivo
Escrever.py

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

- momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/banco de dados\$ python ler.py
(19,)
- momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/banco de dados\$ █

Banco de Dados - Exercício de implementação

Façam um programa para ler toda a base de dados e também inserir os seguintes dados:

Nome	Matrícula	idade	Ingresso	Situação
João	10114385	28	2022	Formado
Pedro	13100001	25	2020	Trancamento
Maísa	11280821	26	2023	Cursando
Patrick	14204123	24	2022	Desistiu

Aplicar filtros para verificar:

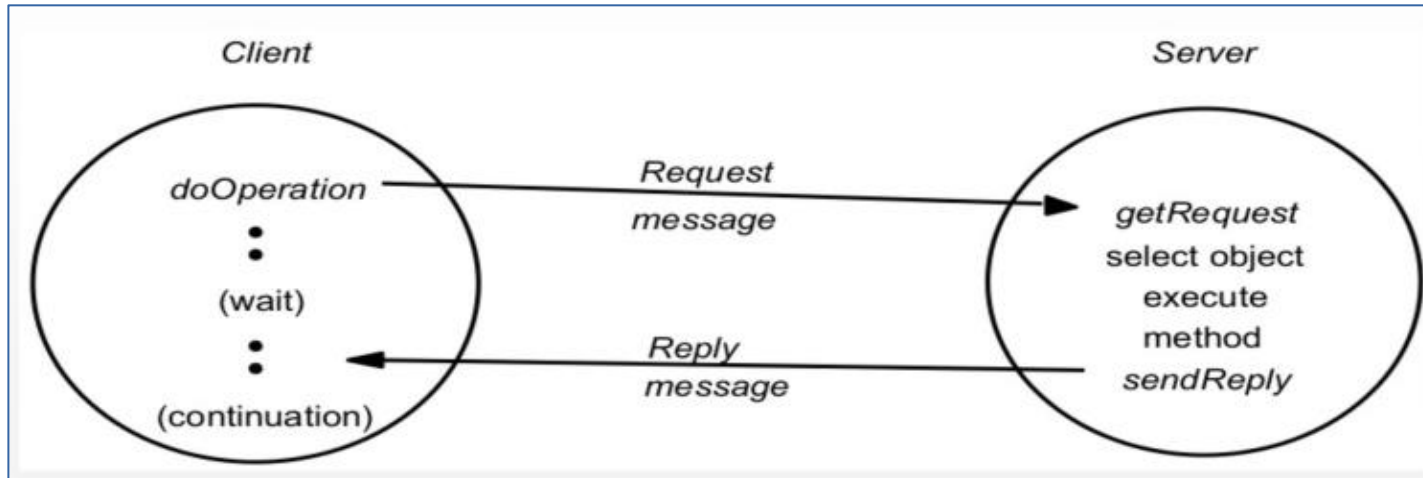
- 1 - Quem trancou o curso?
- 2 - Quem entrou após 2012?
- 3 - Qual a idade de Patrick?
- 4 - Alterar o nome de Pedro para Carlos (UPDATE)

Sumário

- Socket
- Websocket
- Banco de dados
- **Invocação Remota de Procedimentos**
- File Transfer
- Rest API AWS Lambda

Invocação Remota de Procedimentos - O que é

Invocação remota é um mecanismo de comunicação que é executado fazendo uso de um middleware entre as aplicações e em cima dos protocolos-base de comunicação UDP e TCP.



Invocação Remota de Procedimentos - Como

- Sendo realizado por um processo de Request-Reply, esta forma de comunicação é projetada para suportar as funções e as trocas de mensagens em interações cliente e servidor típicas.
- No caso normal, a comunicação por requisição-resposta é síncrona, pois o processo cliente é bloqueado até que a resposta do servidor chegue. Ela também é confiável, pois a resposta do servidor é efetivamente uma confirmação para o cliente.
- A comunicação por requisição-resposta assíncrona é uma alternativa útil em situações em que os clientes podem recuperar as respostas posteriormente.

Invocação Remota - Exercício de implementação

Server.py

```
1 from xmlrpc.server import SimpleXMLRPCServer
2 from xmlrpc.server import SimpleXMLRPCRequestHandler
3
4 class RequestHandler(SimpleXMLRPCRequestHandler):
5     rpc_paths = ('/RPC2',)
6
7 with SimpleXMLRPCServer(('localhost', 8005),
8     requestHandler=RequestHandler) as server:
9     server.register_introspection_functions()
10
11     def adder_function(x, y):
12         print(x+y)
13         return x + y
14
15     server.register_function(adder_function, 'add')
16
17 server.serve_forever()
```

Saída no servidor

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
o momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/rmi$ python servidor.py
25
127.0.0.1 - - [13/Apr/2023 10:56:40] "POST /RPC2 HTTP/1.1" 200 -
```

Client.py

```
1 import xmlrpc.client
2 s = xmlrpc.client.ServerProxy('http://localhost:8005')
3
4 numero1 = input("1o numero: ")
5 numero2 = input("2o numero: ")
6
7 print('Somar: ', s.add(int(numero1), int(numero2)))
```

Entrada/Saída no cliente

```
o momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/rmi$ python cliente.py
1o numero: 15
2o numero: 10
Somar: 25
o momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/rmi$
```

Invocação Remota - Exercício de Implementação

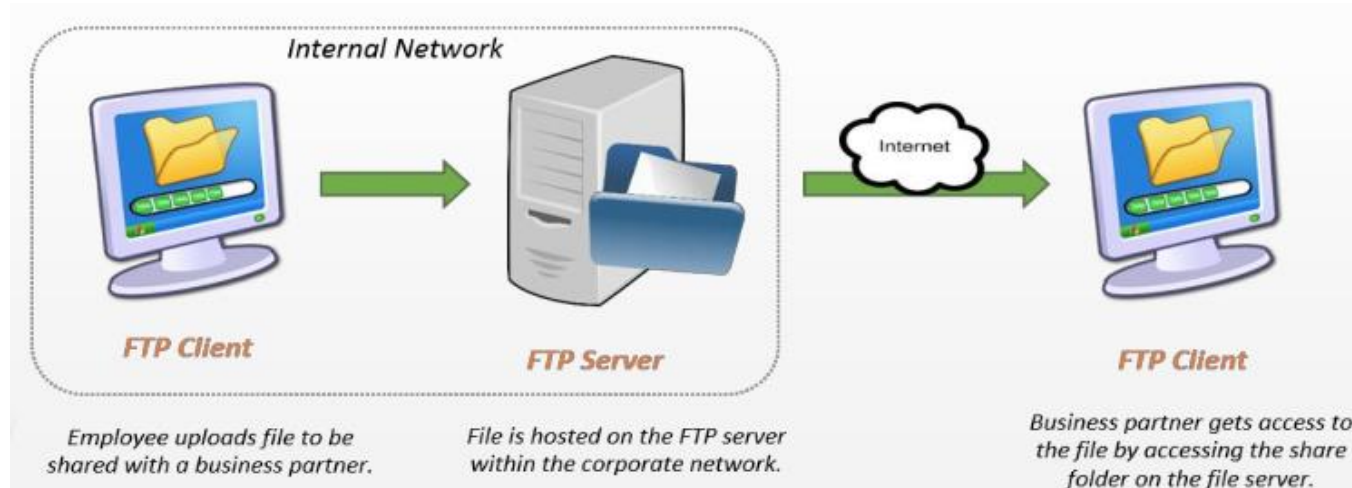
- A partir do código do exemplo, criar um servidor capaz de receber dois números, somar, multiplicar, elevar à potência, dividir e subtrair estes dois números.
- Por fim, enviar de volta os valores calculados ao cliente e apresentar no console os resultados.

Sumário

- Socket
- Websocket
- Banco de dados
- Invocação Remota de Procedimentos
- **File Transfer**
- Rest API AWS Lambda

File Transfer - O que é

Transferência de arquivo é a transmissão de um arquivo de computador através de um canal de comunicação de um sistema computacional para outro. Tipicamente, a transferência de arquivos é mediada através de protocolos de comunicação, sendo o mais comum o File Transfer Protocol (FTP).



File Transfer - Como funciona

- No processo de transferência e recebimento de arquivos pela internet, o FTP funciona em torno de dois protagonistas: o cliente e o servidor.
- O computador que atua como cliente consegue acesso aos arquivos (hospedados na Internet ou localmente) através de um programa que se conecta ao computador que atua como servidor. Já o computador que atua como servidor geralmente possui programas (basicamente uma interface/API) disponíveis para permitir a conexão de computadores externos a ele.
- Essa operação precisa ser segura. Por isso, ela sempre pede alguma autenticação para proteger as transferências de dados. Ou seja, é obrigatório ter um login e uma senha de acesso para transferir arquivos pelo FTP.

File Transfer - Como funciona

Resumo básico passo a passo do que acontece ao usar o FTP.

- Você inicia um programa de FTP no seu computador que atua como cliente;
- Você insere um usuário e senha de acesso no programa de FTP;
- O servidor recebe o pedido de conexão, reconhece os dados e redireciona o seu acesso para o diretório onde estão os arquivos;
- Você já fez o intercâmbio de dados, transferindo arquivos do seu computador para o servidor e vice-versa;
- Depois de realizar todas as tarefas, a conexão entre computador e servidor é encerrada.

File Transfer - Exercício de implementação

Server.py

```
1 import socket
2 port = 6001
3 s = socket.socket()
4 host = 'localhost'
5 s.bind((host, port))
6 s.listen(1)
7
8 while True:
9     conn, addr = s.accept()
10    print ('Conectado por:', addr)
11
12    filename='teste.xlsx'
13    f = open(filename,'rb')
14    l = f.read(1024*16)
15
16    while (l):
17        conn.send(l)
18        l = f.read(1024*16)
19
20    f.close()
21
22    print('Envio completo')
23    conn.close()
24    print('Conexão fechada')
```

Client.py

```
1 import socket
2 s = socket.socket()
3 host = 'localhost'
4 port = 6001
5
6 s.connect((host, port))
7
8 with open('teste.xlsx', 'wb') as f:
9     print ('Arquivo Aberto')
10
11     while True:
12         print('Recebendo dados...')
13         data = s.recv(1024*16)
14
15         if not data:
16             break
17         f.write(data)
18
19     f.close()
20     print('Transferência realizada com sucesso')
21     s.close()
22     print('Conexão fechada')
```

File Transfer - Exercício de implementação

Exemplo saída de execução

Saída no servidor

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  
momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/filefer$ python servidor.py  
Conectado por: ('127.0.0.1', 50308)  
Envio completo  
Conexão fechada  
█
```

Saída no cliente

```
momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/filsfer$ python cliente.py  
Arquivo Aberto  
Recebendo dados...  
Transferência realizada com sucesso  
Conexão fechada  
○ momo@momo-Inspiron-15-3567:~/Área de Trabalho/python SI/file transfer$ █
```

File Transfer - Exercício de implementação

- A partir dos códigos do exemplo, editar o arquivo “FT_arquivo_recebido.xlsx” e desenvolver um servidor e um cliente para enviar de volta, sobrescrevendo o “FT_teste.xlsx” antigo, salvando a nova versão.
- Desafio: Escrever um único código para o servidor e um único código para o cliente para fazer essa troca de arquivos, com um tempo entre enviar/receber para poder editar o arquivo .xlsx

Sumário

- Socket
- Websocket
- Banco de dados
- Invocação Remota de Procedimentos
- File Transfer
- **Rest API AWS Lambda**

API AWS Lambda

Baseado na Rest API Calc, documentada ao lado,
faça os testes com o Postman nas seguintes rotas
<https://mrl0vy3w26.execute-api.sa-east-1.amazonaws.com>
<https://mrl0vy3w26.execute-api.sa-east-1.amazonaws.com>
<https://mrl0vy3w26.execute-api.sa-east-1.amazonaws.com>
<https://mrl0vy3w26.execute-api.sa-east-1.amazonaws.com/api/mult/4/5>
<https://mrl0vy3w26.execute-api.sa-east-1.amazonaws.com/api/div/4/5>



API Gateway: [calc](#)

arn:aws:execute-api:sa-east-1:007901780548:mrl0vy3w26/*

API endpoint: <https://mrl0vy3w26.execute-api.sa-east-1.amazonaws.com/api/>

▼ Detalhes

API type: **REST**

Authorization: **NONE**

Binary media types: **application/octet-stream, application/x-tar, application/zip, audio/basic, audio/ogg, audio/mp4, audio/mpeg, audio/wav, audio/webm, image/png, image/jpg, image/jpeg, image/gif, video/ogg, video/mpeg, video/webm**

isComplexStatement: **Não**

Method: **GET**

Resource path: **/**

Service principal: **apigateway.amazonaws.com**

Stage: **api**

Statement ID: **b03bbfd5-9422-4d88-8586-b533ee9c6f4e**

Saídas do Postman

https://mr10vy3w26.execute-api.sa-east-1.amazonaws.com/api/soma/4/5

GET https://mr10vy3w26.execute-api.sa-east-1.amazonaws.com/api/soma/4/5

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Key	Value	Bulk Edit
Key	Value	

Body 200 OK 467 ms 544 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2 "A soma: ": 9
3
```

https://mr10vy3w26.execute-api.sa-east-1.amazonaws.com/api/sub/4/5

GET https://mr10vy3w26.execute-api.sa-east-1.amazonaws.com/api/sub/4/5

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Key	Value	Bulk Edit
Key	Value	

Body 200 OK 59 ms 559 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2 "A subtração: ": -1
3
```

https://mr10vy3w26.execute-api.sa-east-1.amazonaws.com/api/mult/4/5

GET https://mr10vy3w26.execute-api.sa-east-1.amazonaws.com/api/mult/4/5

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Key	Value	Bulk Edit
Key	Value	

Body 200 OK 75 ms 562 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2 "A multiplicação": 20
3
```

https://mr10vy3w26.execute-api.sa-east-1.amazonaws.com/api/div/4/5

GET https://mr10vy3w26.execute-api.sa-east-1.amazonaws.com/api/div/4/5

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

Key	Value	Bulk Edit
Key	Value	

Body 200 OK 62 ms 552 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2 "A Divisão": 0.8
3
```


APIS na AWS Lambda - Exercício de implementação

Implementar um cliente Chalice em Python para executar as quatro operações da Rest API Calc publicada na AWS Lambda (somar, subtrair, dividir e multiplicar)

As operações recebem três parâmetros

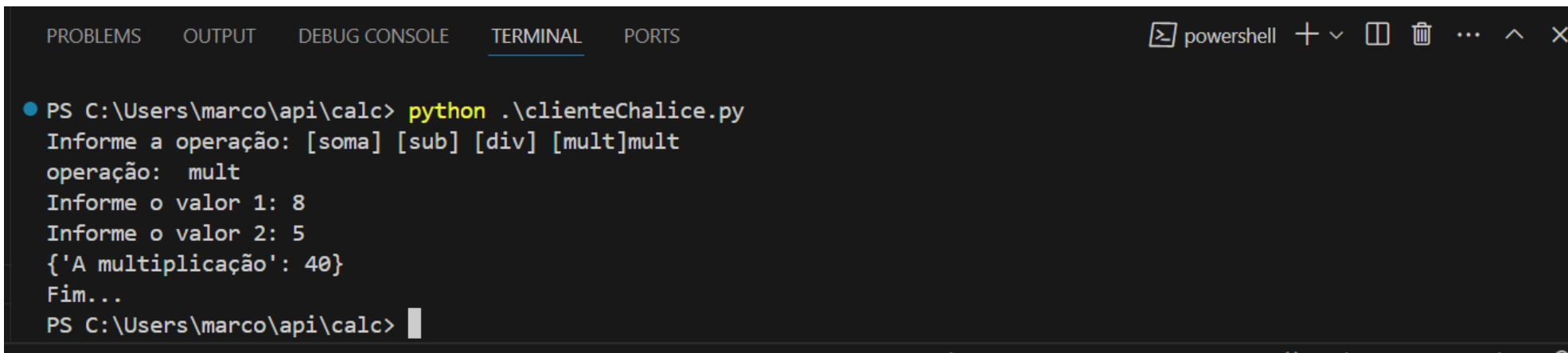
Operação

Valor1

Valor2

E retorna um JSON de resultado

Um exemplo de execução, via terminal:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + - [ ] [X] ... ^ X

● PS C:\Users\marco\api\calc> python .\clienteChalice.py
Informe a operação: [soma] [sub] [div] [mult]mult
operação: mult
Informe o valor 1: 8
Informe o valor 2: 5
{'A multiplicação': 40}
Fim...
PS C:\Users\marco\api\calc>
```

Mecanismos de Comunicação em Sistemas Distribuídos

Prof. Marcos Momo
Prof. Ricardo J. Rabelo

abril/2024