LABORATÓRIO PRÁTICO

Objetivo: Chat com FastStream e RabbitMQ

Pré-requisitos para o Laboratório

1. Iniciar nova instância EC2 na AWS (Debian) e acessá-la via SSH
2. Liberar porta de entrada TCP/8080, TCP/15672 e TCP/5672 no console de administração
3. Instalar o gerenciador de ambientes/pacotes uv (via script oficial - site do uv)
4. Instalar o VS Code Server (via script oficial - site do code-server)
5. Altere o socket de escuta padrão do Code Server

```
sudo nano ~/.config/code-server/config.yaml
bind-addr: 0.0.0.0:8080
<<Aproveite e já altere e/ou copie a senha de acesso contida neste arquivo>>
```

6. Configure um ambiente virtual isolado, e inicie o code-server…

```
uv init codeserver
cd codeserver
uv venv
source .venv/bin/activate
uv pip install --upgrade pip
uv pip install "faststream[cli]"
uv pip install "faststream[rabbit]"
uv pip install uvicorn
code-server
```

7. Acesse seu novo ambiente para desenvolvimento no server-side: http://ip_server:8080

8. Crie o console para gerenciar as salas de bate-papo…

```
from faststream.rabbit import RabbitBroker
from faststream import FastStream
import uuid

broker = RabbitBroker("amqp://guest:guest@ip_server:5672//")
app = FastStream(broker)

salas = {}
clientes = {}

@app.subscriber("salas")
async def gerenciar(req):
    action = req.get("action")

    if action == "criar":
```

```
        sala_id = str(uuid.uuid4())[:6]
        fila = f"sala.{sala_id}"
        salas[sala_id] = fila
        return {"ok": True, "sala_id": sala_id, "fila": fila}

    if action == "entrar":
        sala_id = req["sala_id"]
        client_id = req["client_id"]

        if sala_id not in salas:
            return {"erro": "Sala inexistente"}

        clientes[client_id] = sala_id
        return {"ok": True, "fila": salas[sala_id]}

    if action == "admin" and req.get("cmd") == "broadcast":
        sala_id = req["sala_id"]
        msg = f"[ADMIN] {req['msg']}"
        await broker.publish(msg, salas[sala_id])
        return {"ok": True}

    if action == "admin" and req.get("cmd") == "pm":
        cid = req["client_id"]
        await broker.publish(f"[ADMIN PM] {req['msg']}", f"private.{cid}")
        return {"ok": True}

    return {"erro": "Ação inválida"}
```

9. Crie o cliente do qual entra e interage com as salas…

```
from faststream.rabbit import RabbitBroker
from faststream import FastStream
import asyncio
import uuid

broker = RabbitBroker("amqp://guest:guest@ip_server:5672//")
client_id = str(uuid.uuid4())[:6]


async def main():
    print(f"Seu ID: {client_id}")

    escolha = input("[1] Criar sala\n
                     [2] Entrar sala existente\n> ")

    if escolha == "1":
        resp = await broker.publish({"action": "criar"}, "salas", rpc=True)
        sala_id = resp["sala_id"]
        sala_fila = resp["fila"]
        print(f"Sala criada! ID: {sala_id}")
```

```
    if escolha == "2":
        sala_id = input("ID da sala: ")
        resp = await broker.publish(
            {"action": "entrar", "sala_id": sala_id, "client_id": client_id},
            "salas",
            rpc=True
        )
        if "erro" in resp:
            print("Erro:", resp["erro"])
            return

        sala_fila = resp["fila"]

    print(f"Conectado à sala {sala_id}")


    async def receber_sala():
        async with broker.subscribe(sala_fila) as sub:
            async for msg in sub:
                print(f"\n{msg}\nVocê: ", end="")

    async def receber_pm():
        async with broker.subscribe(f"private.{client_id}") as sub:
            async for msg in sub:
                print(f"\n{msg}")
                if "desconectado" in msg or "apagada" in msg:
                    print("\nEncerrando cliente…")
                    exit()

    asyncio.create_task(receber_sala())
    asyncio.create_task(receber_pm())

    while True:
        msg = input("Você: ")
        if msg == "/exit":
            break

        await broker.publish(f"[{client_id}] {msg}", sala_fila)


if __name__ == "__main__":
    asyncio.run(main())
```

10. Crie o menu de comandos do admin com a sala de bate-papo…

```
from faststream.rabbit import RabbitBroker
from faststream import FastStream
import asyncio

broker = RabbitBroker("amqp://guest:guest@ip_server:5672//")

async def send(cmd):
```

```
    await broker.publish(cmd, "salas")

async def main():
    while True:
        print("\n=== ADMIN ===")
        print("1) Broadcast sala")
        print("2) Enviar PM")
        print("3) Kick cliente")
        print("4) Kick ALL")
        print("5) Apagar sala")
        op = input("> ")

        if op == "1":
            sala = input("Sala ID: ")
            msg = input("Mensagem: ")
            await send({"action": "admin", "cmd": "broadcast", "sala_id": sala, "msg": msg})

        elif op == "2":
            cid = input("Client ID: ")
            msg = input("Mensagem: ")
            await send({"action": "admin", "cmd": "pm", "client_id": cid, "msg": msg})

        elif op == "3":
            cid = input("Client ID: ")
            await send({"action": "admin", "cmd": "kick", "client_id": cid})

        elif op == "4":
            sala = input("Sala ID: ")
            await send({"action": "admin", "cmd": "kick_all", "sala_id": sala})

        elif op == "5":
            sala = input("Sala ID: ")
            await send({"action": "admin", "cmd": "apagar_sala", "sala_id": sala})
```

11. Implementar Kick (Expulsar Cliente). publique a mensagem "[ADMIN] Você foi desconectado." na fila privada do cliente (private.{cid}). Após a notificação, o cliente deve ser removido do dicionário de clientes.

Sugestão de comando: {"action": "admin", "cmd": "kick", "client_id": cid}

12. Implementar Kick All (Expulsar Todos da Sala). para cada cliente que pertença à sala_id especificada, publique a mensagem "[ADMIN] Sala resetada. Todos removidos." na fila privada do cliente. Em seguida, remova o cliente do dicionário clientes.

Sugestão de comando: {"action": "admin", "cmd": "kick_all", "sala_id": sala_id}

13. Implementar Apagar Sala. Você pode usar a lógica do Kick All para notificar e remover todos os clientes da sala. Após a remoção dos clientes, a própria sala deve ser removida do dicionário de salas.

Sugestão de comando: {"action": "admin", "cmd": "apagar_sala", "sala_id": sala_id}

14. Crie um Novo Recurso Listar Clientes (Análise de Estado).
    a. Crie um novo comando administrativo "cmd": "clientes_sala" que aceita um sala_id. O comando deve retornar uma lista dos client_id atualmente conectados àquela sala_id.

    b. Por que você deve usar list(clientes.items()) ou uma cópia do dicionário clientes (ou uma estrutura de dados serializável, como uma lista de IDs) antes de enviá-lo de volta ao Admin? Explique a importância da serialização nesse contexto.

13. No código implementado na questão 12 (kick_all), você iterou sobre o dicionário de clientes. Por que é crucial iterar sobre uma cópia (usando list(clientes.items()):) quando você está deletando itens dentro do loop do dicionário original? (Conceito de Mutação durante Iteração).

14. Se você introduzisse um time.sleep(5) dentro de um dos subscribers do Gerenciador (@app.subscriber), isso bloqueia a execução do broker inteiro? Como o FastStream/Asyncio gerencia as tarefas concorrentes para mitigar esse bloqueio.

15. Altere o código do Cliente (questão 9) para usar um timeout RPC de 5 segundos na chamada broker.publish (rpc_timeout=5).

16. Usando um bloco try...except na chamada RPC do cliente, desenvolva a lógica para:
    a. Capturar e identificar a exceção de Timeout (rede/latência).
    b. Capturar e identificar a exceção de Erro Interno (falha não tratada no servidor).