LABORATÓRIO PRÁTICO WEB SOCKETS

Pré-requisitos para o Laboratório

1. Acesso à sua Instância AWS (via VS Code Server ou SSH).
2. Python 3.7+ instalado.
3. Ambiente virtual UV com websockets adicionado
4. Porta 8000 liberada

1. Crie um arquivo server.py. Este código será responsável por manter qual é a cor atual do site e avisar a todos os clientes quando ela mudar.

Transcreva o código abaixo:

```
import asyncio
import websockets

COR_ATUAL = "white"
# Lista de clientes conectados
CLIENTES = set()

async def manipulador(websocket):
    global COR_ATUAL
    # Registra nova conexão
    CLIENTES.add(websocket)
    print(f"Cliente conectado. Total: {len(CLIENTES)}")

    try:
        # Sincronização Inicial: Envia a cor atual assim que entra
        await websocket.send(COR_ATUAL)

        # Loop de Escuta (Mantém a conexão aberta)
        async for message in websocket:
            COR_ATUAL = message

            # Broadcast: Envia a nova cor para todos os conectados
            websockets.broadcast(CLIENTES, COR_ATUAL)

    except websockets.exceptions.ConnectionClosed:
        pass
    finally:
        CLIENTES.remove(websocket)

async def main():
    async with websockets.serve(manipulador, "0.0.0.0", 8000):
        await asyncio.Future() # Mantém rodando para sempre
```

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

Reflita...
Observe a variável CLIENTES. Por que ela é necessária? Se o Cliente A mandar uma mensagem, como o Cliente B ficaria sabendo se não existisse essa lista? No modelo HTTP tradicional, o servidor precisaria "lembrar" de quem está acessando o site?

2. Crie um arquivo chamado index.html na mesma pasta. Este arquivo conterá a interface e a lógica JavaScript para abrir o canal de comunicação. Edite a linha de conexão com o IP Público da sua máquina AWS.

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Lab WebSocket</title>
    <style>
        body {
            font-family: sans-serif; transition: background-color 0.5s;
            display: flex; flex-direction: column;
            align-items: center; justify-content: center;
            height: 100vh; margin: 0;
        }
        .botoes { display: flex; gap: 20px; }
        button {
            padding: 20px 40px; font-size: 18px; cursor: pointer;
            border: none; color: white; font-weight: bold; border-radius: 8px;
        }
        #btn-red { background-color: #e74c3c; }
        #btn-green { background-color: #2ecc71; }
        #btn-blue { background-color: #3498db; }
    </style>
</head>
<body>
    <h1>Controle de Cor</h1>
    <div class="botoes">
        <button id="btn-red" onclick="mudarCor('red')">Vermelho</button>
        <button id="btn-green" onclick="mudarCor('green')">Verde</button>
        <button id="btn-blue" onclick="mudarCor('blue')">Azul</button>
    </div>

    <script>
        // Handshake: Conecta ao servidor WebSocket, Substitua pelo endereço IP Público da instância
        const socket = new WebSocket("ws://SEUIP:8000");
```

```
        // Evento: Recebendo dados do servidor (Server-Push)
        socket.onmessage = function(event) {
            const novaCor = event.data;
            document.body.style.backgroundColor = novaCor;
        };

        // Ação: Enviando dados para o servidor
        function mudarCor(cor) {
            socket.send(cor);
        }
    </script>
</body>
</html>
```

Reflita...
Observe a função mudarCor. Ela muda a cor da tela do próprio usuário? Por que essa distinção é importante para garantir que todos vejam a mesma coisa?

3. Teste

1. Instale o Nginx:

   sudo apt update

   sudo apt install nginx

2. No terminal da sua instância, copie o arquivo index.html para a pasta pública do Nginx: sudo cp index.html /var/www/html/index.html
3. Execute o server.py.
4. Acesse o IP em quantas telas quiser (pelo menos 2).
5. Clique em um botão na Tela 1. Observe a Tela 2.

   Analise o comportamento...
   Se você fechar o servidor (Ctrl+C no terminal) e tentar clicar nos botões, o que acontece? A cor muda? Por que isso difere de um site estático comum?

4. Não se contente com a implementação funcional, revise o código, entenda e modifique para testar o WebSockets.

5. Desafio Extra: Balada

Agora vamos testar a capacidade do servidor de agir por conta própria (*Server Push* ativo).

Objetivo: Criar uma funcionalidade onde o servidor muda as cores aleatoriamente a cada 0.5 segundos, transformando todas as telas conectadas em uma festa, sem que ninguém precise clicar em nada.
Precisa alterar algo no cliente?