Live Chat Usando Python

Grupo: Lucas Kumagae, Gabriel de Lima, Henrique Szlachta

Requisitos

RF1: Assim que um usuário entrar no chat, uma mensagem informando o nome do usuário que acabou de se conectar deverá ser enviada para todos os usuários já conectados.

RF2: O usuário poderá digitar o comando "/sair" pra sair do chat.

RF3: O usuário poderá digitar "/[nome_do_usuário]" para enviar mensagens privadas para algum usuário.

RF4: O usuário poderá digitar "/clientes" para mostrar a lista de clientes conectados no chat.

RF5: O usuário deve receber as mensagens de outros usuários em tempo real.

RF6: Assim que um usuário sair do chat, uma mensagem informando o nome do usuário que saiu deverá ser enviada para todos os usuários conectados.

RF7: Caso o cliente perca o contato com o servidor, o sistema deverá exibir uma mensagem de erro.

RF8: Caso o servidor não consiga receber as mensagens de um cliente, o sistema deverá exibir uma mensagem de erro.

RF9: Caso o destinatário especificado na mensagem não seja encontrado na lista de usuários conectados, o sistema deverá exibir uma mensagem de erro.

RF10: Assim que o chat é iniciado, o sistema deverá exigir que o usuário insira seu nome, que será exibido em suas mensagens.

RF11: O sistema não deve permitir que o usuário insira um nome contendo espaços.

Relatório

Sockets

O socket é utilizado para criar um cliente de chat que se conecta com o servidor, podendo enviar e receber mensagens

- `sock.AF_INET`: Define o tipo de endereço IP como IPv4.
- `sock.SOCK_STREAM`: Define o protocolo de transporte como TCP, que é confiável e orientado a conexão.
- `sock_server.bind((HOST, PORTA))`: Associa o socket do servidor ao IP e porta definidos.

- `sock_server.listen()`: Coloca o socket no modo de escuta, permitindo que ele aceite conexões de clientes.
- `sock_conn`: Um novo socket específico para a conexão com o cliente. Esse socket é exclusivo para o cliente que acabou de se conectar.
- `ender`: Endereço IP e porta do cliente.
- `sock_cliente.recv(50)`: Recebe o nome do cliente (primeira mensagem enviada ao se conectar), que é armazenado na lista `lista_clientes`.
- `broadcast(f"{nome} entrou no servidor", "")`: Informa a todos os clientes conectados que um novo cliente entrou.
- `cliente.sendall(f"{remetente} >> {mensagem}".encode())`: Envia a mensagem para cada cliente da lista `lista_clientes`, exceto para o remetente.
- `remover(nome)`: Remove clientes desconectados, se ocorrer um erro ao enviar a mensagem.
- `lista_clientes[destinatario].send(...)`: Envia a mensagem privada para o destinatário especificado, caso ele esteja na lista de clientes conectados.
- `lista_clientes[nome].close()`: Fecha o socket do cliente para liberar os recursos da conexão.
- `sock.socket(sock.AF_INET, sock.SOCK_STREAM)`: Cria um socket do tipo IPv4 (AF_INET) e baseado no protocolo TCP (SOCK_STREAM), que oferece uma conexão confiável.
- `socket_cliente.connect((HOST, PORTA))`: Conecta o cliente ao servidor no endereço IP `HOST` e porta `PORTA`.
- `socket_cliente.sendall(username.encode())`, que transforma a string do nome em bytes para ser enviada.
- `sock_cliente.recv(1024)`: Recebe dados de até 1024 bytes do servidor, que representam uma mensagem.
- `mensagem.decode()`: Decodifica os bytes recebidos para uma string legível.
- `socket_cliente.sendall(mensagem.encode())`: Envia a mensagem ao servidor em formato de bytes, permitindo que todos os clientes conectados (ou destinatários específicos, se for o caso) a recebam.

Threads

No servidor, a cada cliente conectado, uma nova thread é criada para gerenciar o recebimento de mensagens. Essas threads são criadas à partir do loop principal assim que um novo cliente se conecta. A lógica que será executada nessas threads está presente na função “recebe_dados”, que cria um loop para receber as mensagens do cliente.

No cliente, outra Thread também é criada para receber as mensagens provenientes do servidor, e sua lógica está presente na função “recebe_dados”. Essa função cria um loop que verifica as mensagens enviadas pelo servidor para o cliente.

Broadcast

O broadcast em um servidor de chat funciona com o servidor mantendo uma lista de todos os clientes conectados. Quando um cliente envia uma mensagem, o servidor a retransmite para todos os outros clientes na lista. E servidor garante que clientes desconectados sejam removidos da lista para evitar erros ao tentar enviar mensagens para eles.