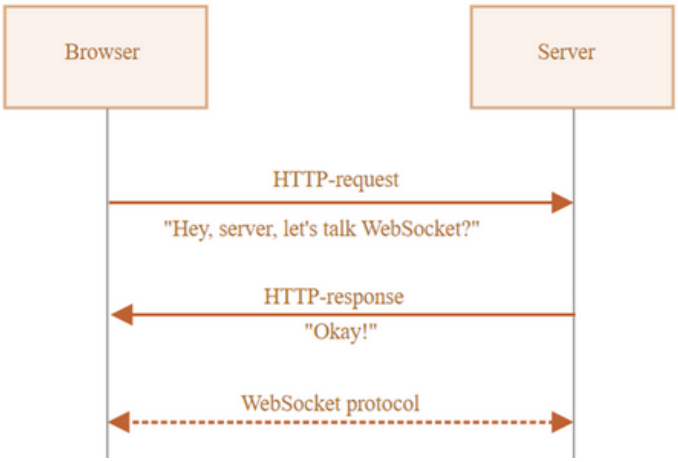
Módulo 19 – Websocket – Atividade 01

Q1) Diga se a seguinte afirmação é verdadeira ou falsa e comprove sua resposta: websocket é um protocolo de rede, assim como o HTTP. Mas ele não especifica o formato das mensagens que podem navegar de um lado a outro pela internet. Ou seja, uma vez que o handshake inicial está feito, servidor e cliente estão livres para trocar pacotes websocket com qualquer formato.

Q2) Diga se a seguinte afirmação é verdadeira ou falsa e comprove sua resposta: Antes de iniciar uma troca de mensagens no protocolo websocket, existe um único ciclo request-response que é necessário para iniciar a conexão, e as mensagens desse ciclo estão no formato HTTP.

Q3) A referência de aula "***Contextualização: introdução a websocket com exemplos no lado do cliente e no lado do servidor***" mostra a seguinte imagem:



Essa imagem mostra 3 fases de uma conexão websocket (cada flecha é uma fase). Sobre essa imagem, diga se a seguinte afirmação é verdadeira ou falsa e explique: Quando um cliente usa o código **let socket = new Websocket("ws://servidor.com")** e depois **socket.send("mensagem do cliente para o servidor")**, esse último código corresponde à primeira flecha da imagem (primeira fase).

Q4) Ainda sobre a mesma imagem, diga se a seguinte afirmação é verdadeira ou falsa e explique: quando um cliente usa o código **socket.onmessage = (event) => console.log("mensagem é: " + event.data)**, isso serve para que o cliente receba a mensagem que vem do servidor na segunda flecha da imagem (segunda fase).

Q5) Diga se a seguinte afirmação é verdadeira ou falsa: Um servidor HTTP pode ao mesmo tempo servir uma API REST em certas rotas, arquivos estáticos em outras rotas, e também permitir conexões websocket em outras rotas.

Q6) Crie um backend e frontend para chat usando websocket.
O frontend pode ser um único HTML (e JS) simples servido pelo backend. Deve ter um espaço para digitar uma mensagem, um botão para enviá-la, e um espaço para mostrar todas as mensagens.

Quando um cliente acessa o frontend, ele estabelece uma conexão websocket com o servidor e passa a receber todas as mensagens que forem publicadas a partir de então (não recebe as mensagens anteriores ao momento da conexão). A cada mensagem recebida (que é uma simples string), o frontend a exibe no HTML no espaço reservado para isso.

Também no momento da conexão do cliente, o servidor cria um `username` único para ele (sugestão: armazene esse `username` no próprio objeto socket que representa o cliente: **socket.username = “nome aleatório aqui”;**) . Esse username deve ser criado e armazenado no backend, não no frontend. Em seguida o servidor envia para todos os outros clientes conectados a mensagem “<username> entrou”. Por exemplo, se o cliente que acaba de entrar recebeu `username` “joaozinho”, a mensagem fica “joaozinho entrou”. Essa mensagem deve ser enviada para todos **exceto** o cliente que está entrando. Para conseguir enviar a mensagem para todos os clientes conectados, você vai precisar armazenar no servidor os sockets de todos os clientes numa variável global. Quando algum cliente sair (evento “close”), lembre-se de retirar o socket dele da lista.

Quando um cliente envia uma mensagem, o servidor recebe essa mensagem e prefixa a ela o `username` do cliente seguido de espaço-em-branco e sinal de “:”. Por exemplo, se a mensagem foi “olá!” e esse cliente tem `username` “joaozinho”, então o servidor reformata a mensagem para “joaozinho: olá!”. Depois o servidor envia a mensagem reformatada para todos os clientes (incluindo o próprio emissor da mensagem original, o joaozinho).

Quando um cliente sai (evento “close” do websocket), o servidor envia a todos os outros a mensagem: “<username> saiu”. Por exemplo, se o cliente que saiu tinha `username` “joaozinho”, a mensagem fica “joaozinho saiu”.

Sugestão: para gerar nomes aleatórios que sejam únicos mas ao mesmo tempo “divertidos”, você poda usar o pacote https://www.npmjs.com/package/unique-names-generator

Requisito: o frontend deve usar o objeto **Websocket** nativo, sem bibliotecas como Socket.io. O backend deve usar a biblioteca **ws** ou **express-ws**, não pode usar outras bibliotecas de websocket como Socket.io.

Envie seu projeto compactado.

Respostas

Q1)

Falsa. A primeira parte é verdadeira, o websocket é um protocolo de rede, assim como o http. No entanto, as mensagens são enviadas no que chamamos de “frames”, fragmentos de dados que podem ser enviados de qualquer lado.

Q2)

Verdadeira. A primeira requisição, que vai estabelecer o handshake, vai enviar uma mensagem em http requisitando a mudança para o protocolo websocket. Caso o servidor responda à requisição concordando com o upgrade para websocket, a troca de mensagens passará a empregar o protocolo ws.

Q3)

A afirmação é falsa. Primeiro é necessário estabelecer a conexão, o que é feito enviando uma requisição http (primeira fase). O servidor vai dar uma resposta, também usando protocolo http

concordando com a solicitação para mudança para ws (segunda fase). Com os dois concordando com a mudança, entramos na terceira fase, onde é estabelecido um túnel bidirecional e as trocas podem ocorrer. Portanto, “socket.send” faz parte da terceira fase, onde já estamos usando o protocolo websocket para enviar uma mensagem.

Q4)

A afirmação é falsa. O primeiro trecho está correto, trata-se de um código empregado para que o cliente receba mensagens do servidor e mostre na tela a mensagem recebida, no entanto, isso já faz parte da conexão via websocket, com o canal de comunicações bidirecional já estabelecido, portanto, faz parte da etapa 3.

Q5)

A afirmação é verdadeira. Para que uma conexão com protocolo seja estabelecida, uma requisição http é enviada primeira e se o servidor concordar com a conexão as trocas passam a ser feitas usando o protocolo websocket.

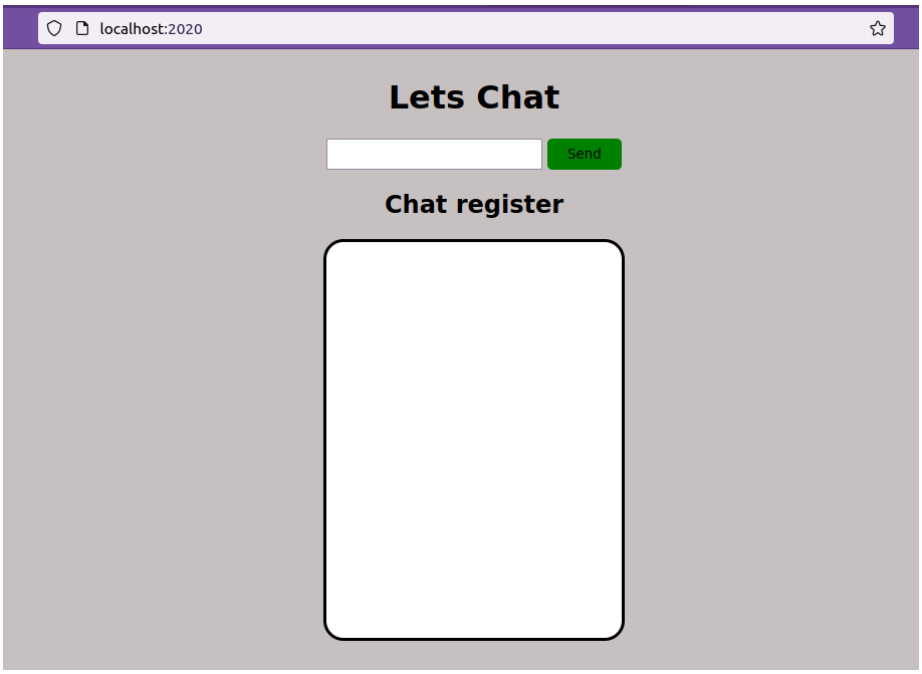
Q6)

O projeto completo foi enviado como anexo. A partir da raiz do projeto, aplicou-se o comando a seguir:

node server.js

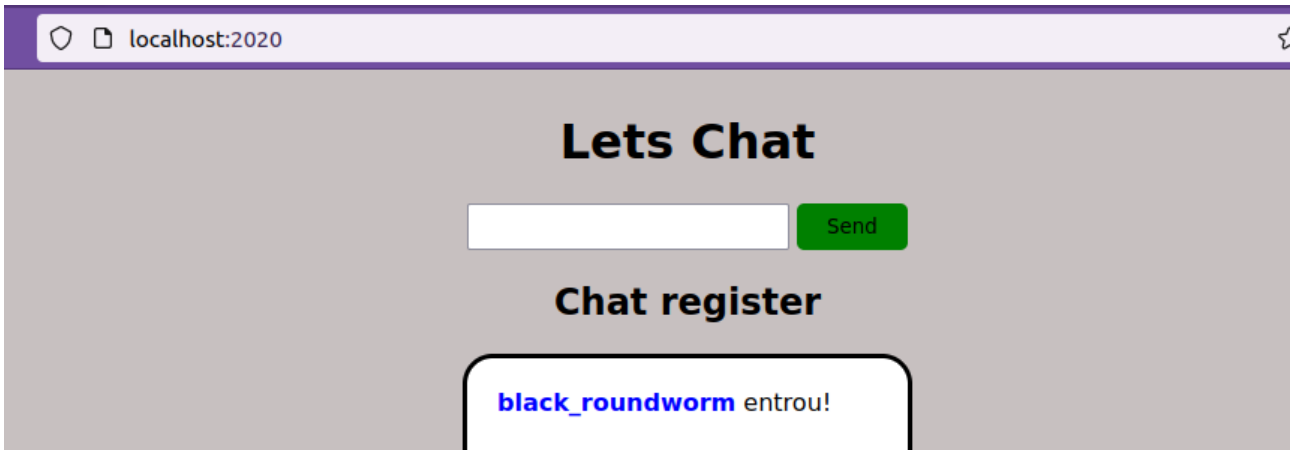
```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
Hard/Alpha-edtechcycle02/Module19_websocket/Aula01/Q6_aula01_chat_websocket$ nod
e server.js
Server is running on http://localhost:2020
```

A partir do navegados, podemos acessar o nosso client com o chat, conforme na imagem abaixo:

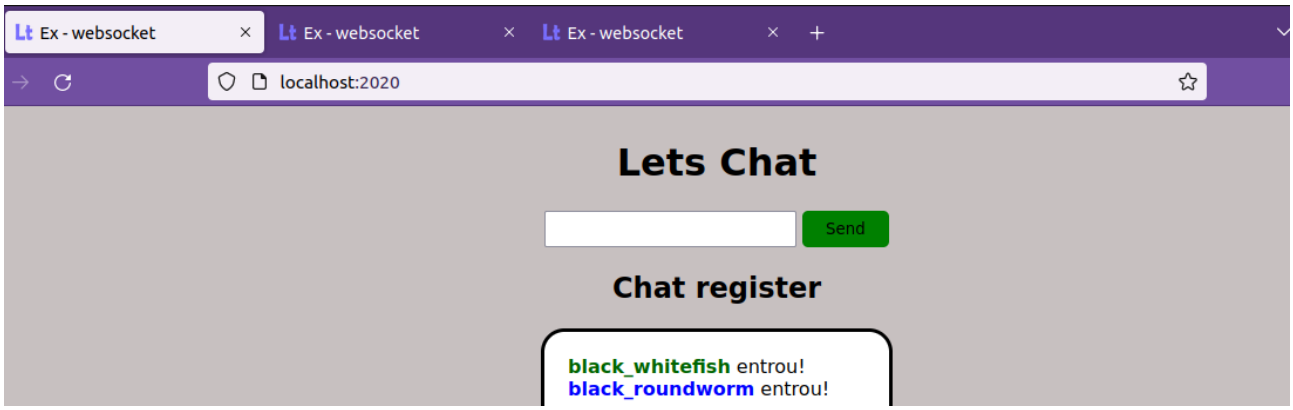


Nota-se que o chat está vazio, pois é nossa primeira janela e, portanto, não tem nenhum aviso. Vamos adicionar uma nova janela, representando nosso segundo client. Ao fazer isso, o primeiro client recebe uma mensagem avisando que um novo usuário se juntou ao chat:

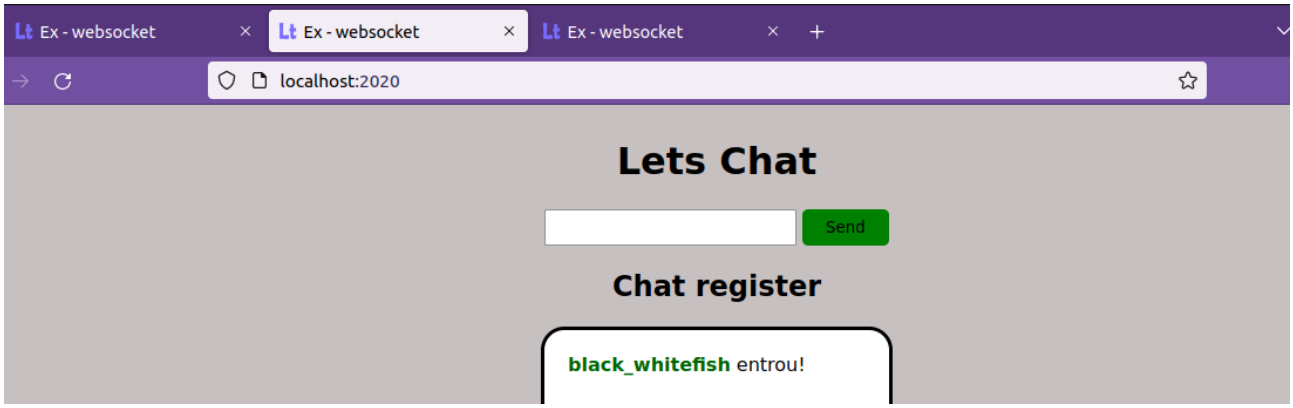
CLIENT 1:



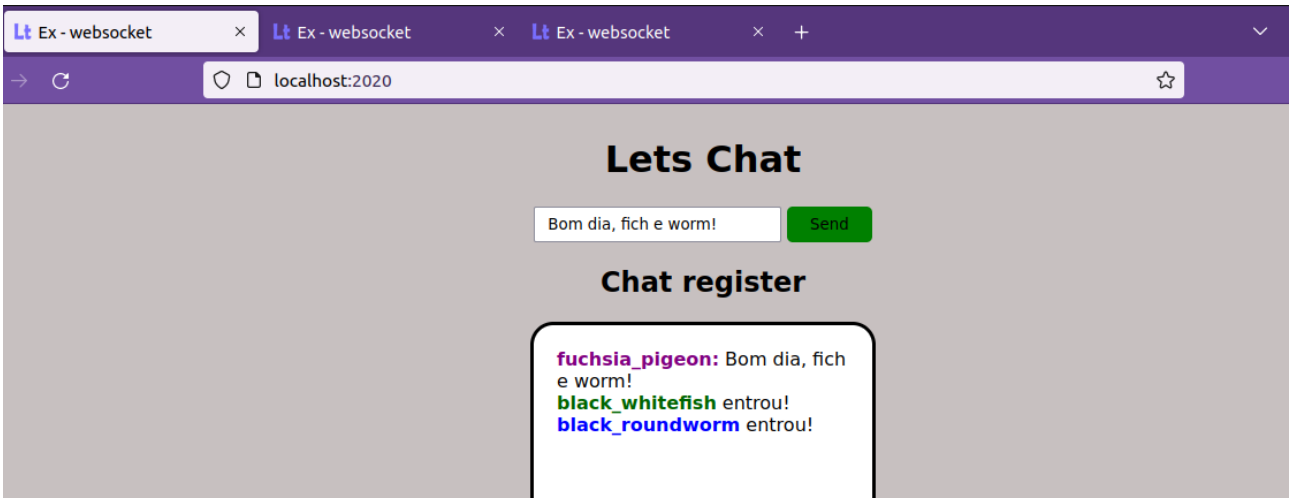
Vamos criar um terceiro usuário. Com o terceiro usuário sendo criado, nota-se que uma nova mensagem é exibida no CLIENT 1:



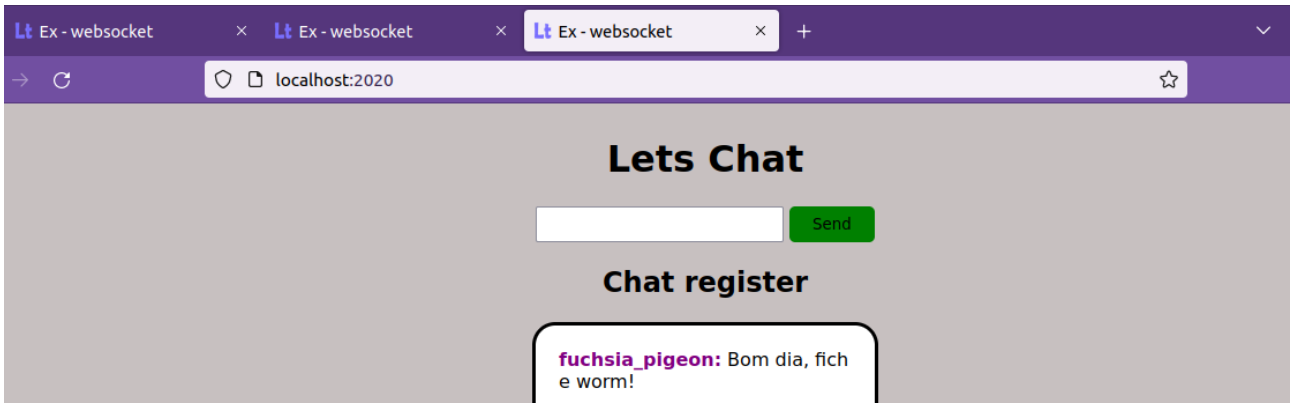
Enquanto isso, o CLIENT 2 está assim:



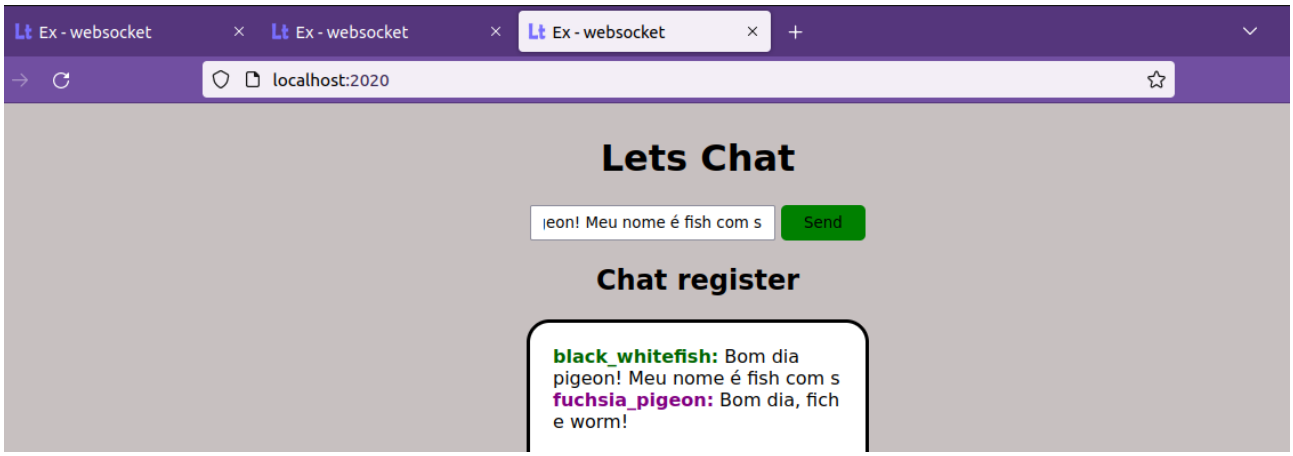
A partir do CLIENT 1, vamos enviar uma mensagem de bom dia:



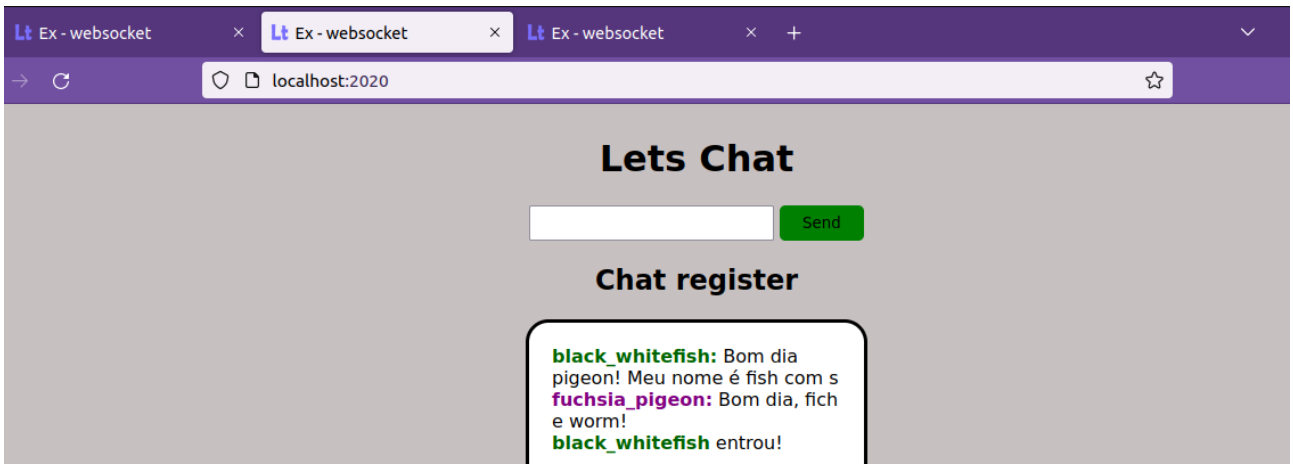
Observando o CLIENT 3, teremos o seguinte:



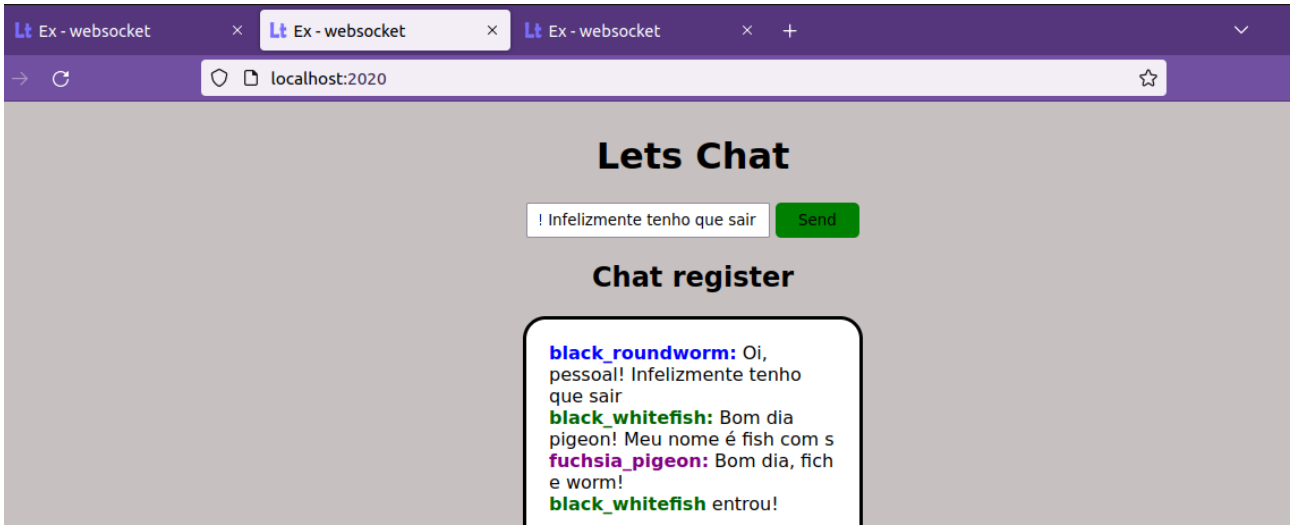
A partir do CLIENT 3, vamos enviar uma mensagem:



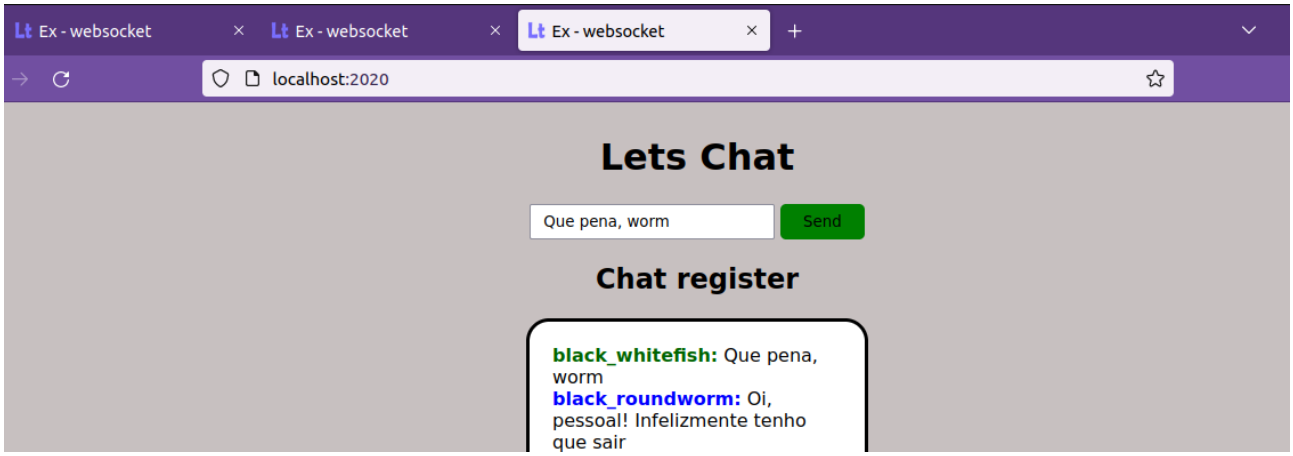
O CLIENT 2 ficou assim:



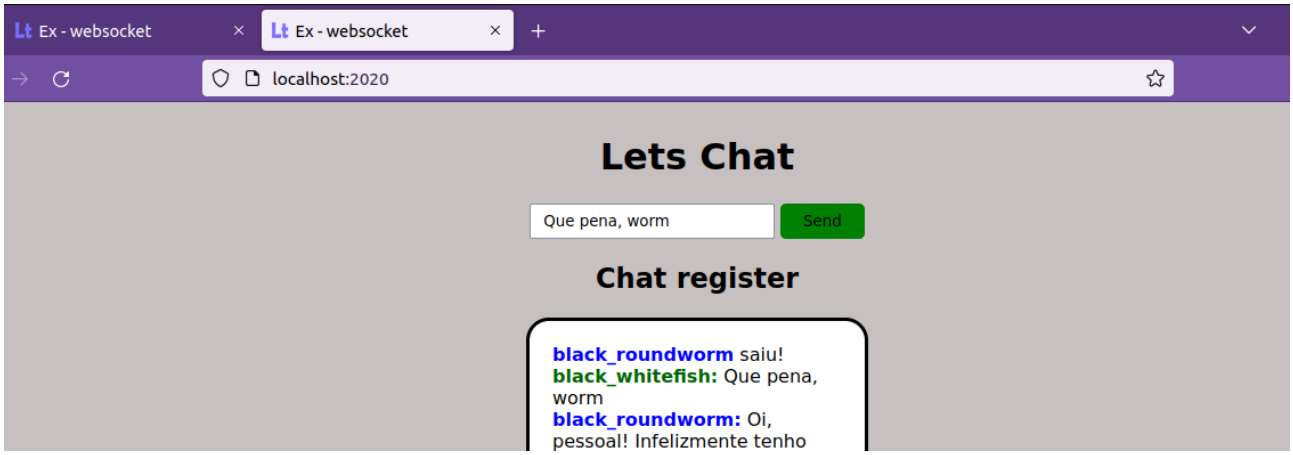
Vamos mandar uma mensagem do CLIENT 2:



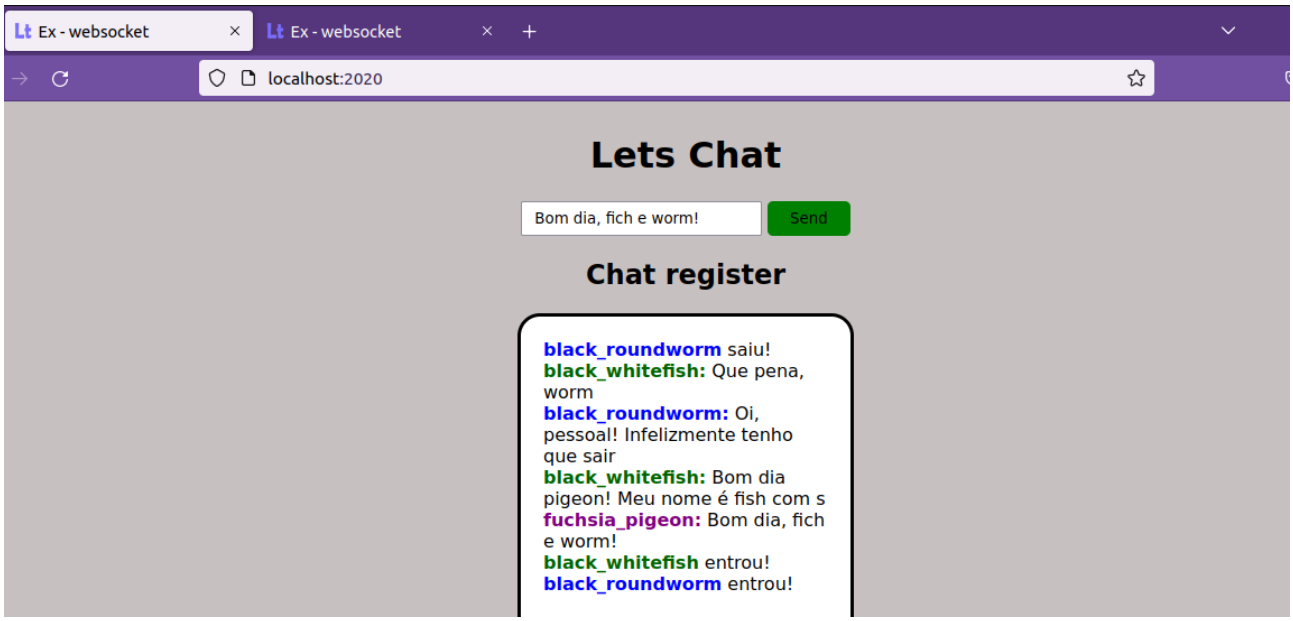
A partir do CLIENT 3, vamos enviar uma mensagem. Logo após a mensagem, vamos fechar a aba do CLIENT 2 (simulando sua saída do chat):



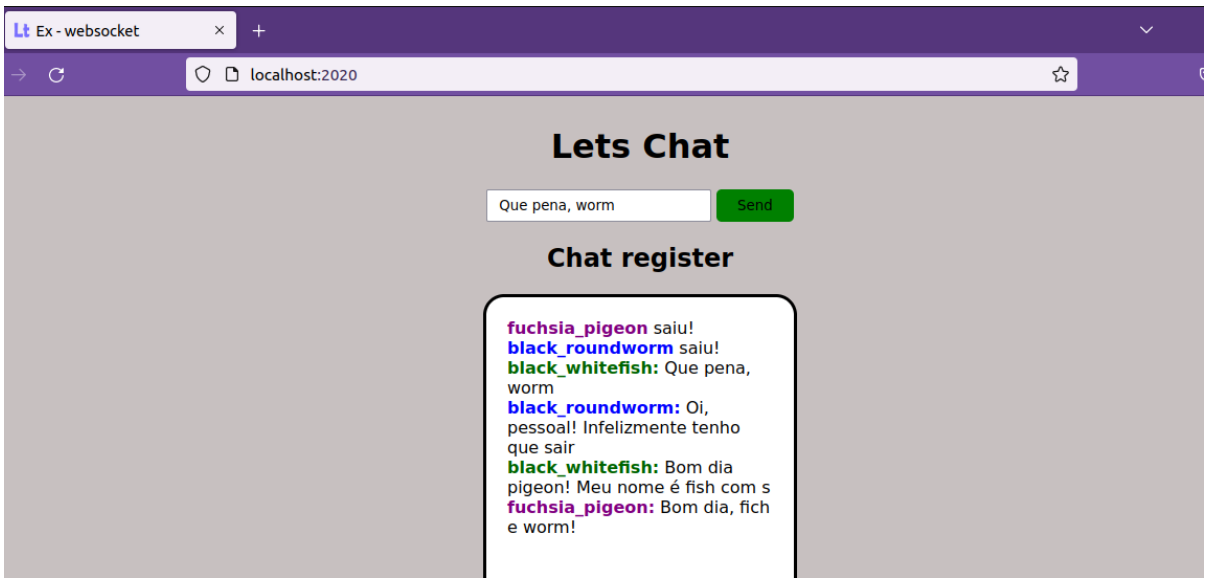
A próxima imagem mostra como ficou o chat do CLIENT 3 após a saída do CLIENT 2 (worm):



No CLIENT 1, temos o seguinte:



Para finalizar, o CLIENT 1 vai sair também. A próxima imagem mostra como ficou o chat do CLIENT 3, após a saída do CLIENT 1 (pigeon):



E dessa forma, finalizamos a atividade.