MANUAL DO PRODUTO

Networker

Chat em tempo real com Spring Boot, WebSocket/STOMP e um servidor TCP textual.

VERSÃO	DATA	BASE
1.0	Agosto de 2026	Spring Boot 2.7.16 · Java 17+

github.com/eduardofabrii/networker

Documento gerado a partir de execução local real do sistema e de capturas de tela em 1440×900 / 390×844.

SUMÁRIO

O que há neste manual

> As páginas de tela seguem sempre a mesma estrutura: título e descrição, captura real anotada com legendas numeradas, cartões de funcionalidades, tabela de campos e uma observação de comportamento.

VISÃO GERAL

Visão geral do produto

O **Networker** é uma aplicação de chat em tempo real construída como projeto acadêmico de redes. Ela combina duas formas de comunicação sobre a mesma lista de participantes: um **servidor TCP textual** (porta 12345), pensado para clientes de terminal, e uma **interface web** que fala WebSocket/STOMP com o mesmo backend Spring Boot (porta 8080).

Para quem é
Estudo de sockets, WebSocket e mensageria. Uso em laboratório/demonstração, não em produção.

O que faz
Mensagens públicas para todos, mensagens privadas (via TCP) e broadcasts operacionais que aparecem na web.

Lista unificada
Clientes TCP e usuários da web aparecem juntos em "Usuários Online".

API de operação
Endpoints REST sob /api/tcp para ligar/desligar o servidor TCP e inspecionar o estado.

EM UMA FRASE POR CAMADA

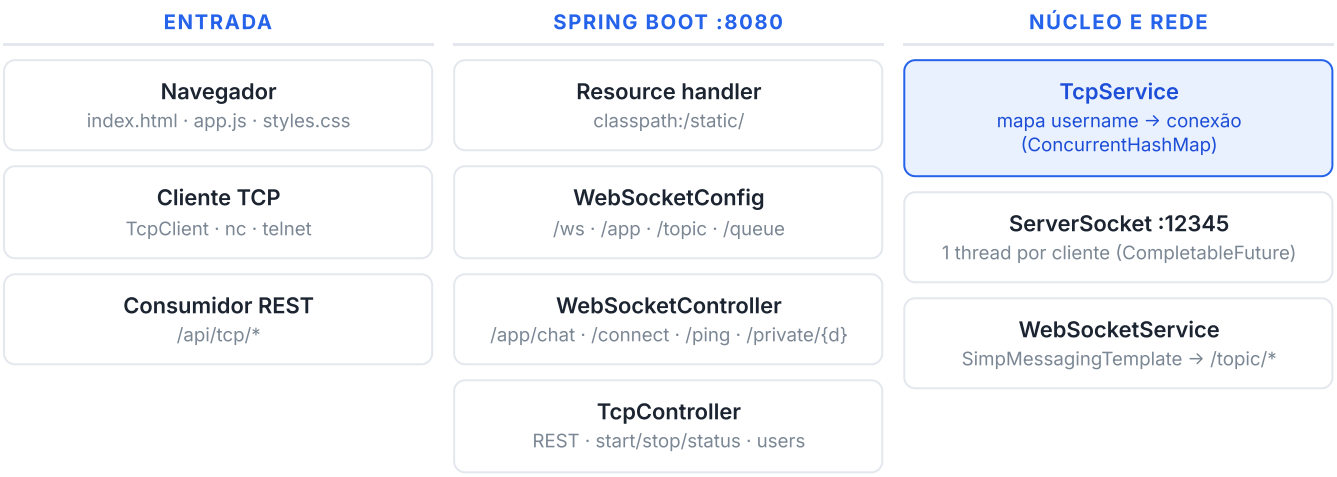
CAMADA	TECNOLOGIA	PAPEL
Interface	HTML + Bootstrap 5 + JS puro	Tela única de chat; sem framework SPA, sem build.
Tempo real	WebSocket + STOMP + SockJS	Entrega de mensagens e da lista de presença ao navegador.
Núcleo	Spring Boot 2.7.16	Controllers REST, serviço de WebSocket e servidor TCP em threads.
Rede bruta	java.net ServerSocket	Protocolo textual orientado a linha na porta 12345.
Persistência	— nenhuma	Todo o estado é em memória; nada sobrevive a um reinício.

OBSERVAÇÃO
Este manual documenta o sistema **como ele se comporta hoje**, incluindo pontos inacabados (tratamento de nome duplicado na web, rota `/` com erro 500). Eles estão sinalizados nas páginas correspondentes.

ARQUITETURA

Arquitetura e componentes

Um único processo Spring Boot atende três frentes ao mesmo tempo: arquivos estáticos, um *broker* STOMP sobre WebSocket e um servidor de sockets TCP. Todos convergem para o `TcpService`, que guarda o mapa de participantes.



PORTAS E ENDPOINTS-CHAVE

RECURSO	VALOR	OBSERVAÇÃO
HTTP (web + API)	`8080`	Interface em `/index.html`; API em `/api/tcp`.
Servidor TCP	`12345`	Inicia parado; suba via `POST /api/tcp/server/start`.
WebSocket	`/ws` (SockJS)	*Broker* simples em memória; canais `/topic/*`.
Limite de mensagem	`8192` bytes	Texto e binário, definido em `application.properties`.

> ⚠ **OBSERVAÇÃO**
> Usuários da web são registrados no `TcpService` com uma "conexão" sem socket. Por isso um `BROADCAST` vindo do TCP percorre esse mapa e, para os usuários web, é entregue pelo caminho do WebSocket.

PRIMEIROS PASSOS

Como executar e acessar

O projeto não depende de banco de dados nem de serviços externos. Requisitos: JDK 17+ (validado também em Java 21) e Maven. As bibliotecas de front (Bootstrap, SockJS, STOMP) são carregadas por CDN, então a máquina precisa de internet ao abrir a tela.

1 · SUBIR O SERVIDOR

```
# na raiz do repositório networker
./mvnw spring-boot:run
# aplicação em http://localhost:8080 — abra /index.html
```

2 · ENTRAR NO CHAT

1. Acesse `http://localhost:8080/index.html`.
2. No modal, aceite o nome sugerido ou digite outro e clique em **Entrar no Chat**.
3. Abra uma segunda aba com outro nome para ver a troca de mensagens e a lista de online.

3 · (OPCIONAL) LIGAR O SERVIDOR TCP

```
curl -X POST "http://localhost:8080/api/tcp/server/start"
# depois, um cliente de terminal:
printf 'LOGIN:ana\nLISTAR_USUARIOS\nBROADCAST:ola\nSAIR\n' | nc localhost 12345
```

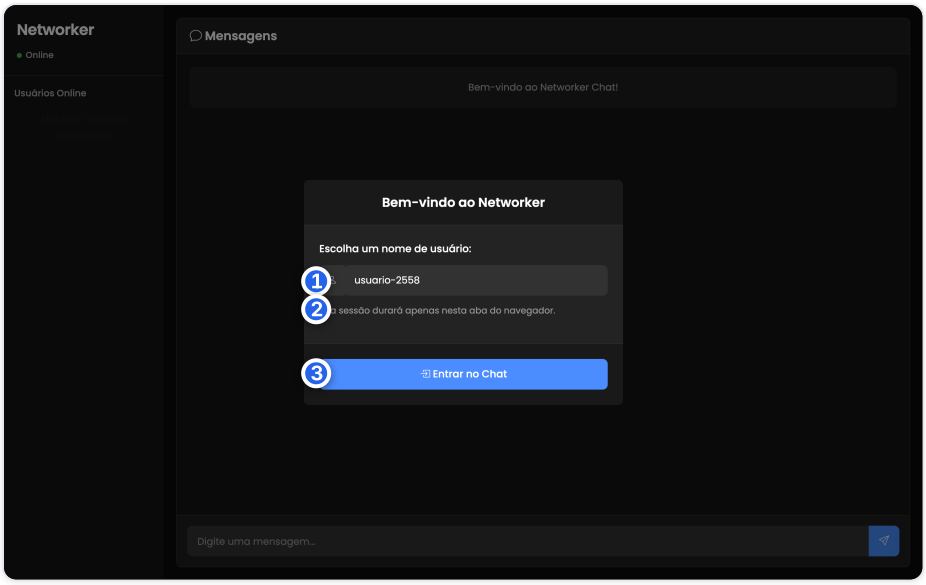
⚠ **OBSERVAÇÃO**

Para encerrar tudo: `Ctrl` + `C` no processo do Maven. O servidor TCP também pode ser desligado sem parar o Spring via `POST /api/tcp/server/stop`.

TELA

Identificação do usuário

Primeira tela ao abrir `/index.html`. Um modal bloqueante pede um nome de exibição antes de liberar o chat. Não há cadastro, senha ou recuperação de conta — a identificação é apenas um rótulo para a sessão atual.



1. Campo **Nome de usuário**, pré-preenchido com um valor aleatório `usuario-1234` gerado no carregamento.
2. Aviso de escopo da sessão: ela vale apenas para aquela aba do navegador.
3. Botão **Entrar no Chat** — única ação da tela; com o campo vazio a borda fica vermelha por 2 s e nada acontece.

Nome sugerido

O campo já vem com `usuario-` + número aleatório (1000–9999), então é possível entrar com um clique.

Validação client-side

Nome vazio (após *trim*) bloqueia a entrada e aplica a classe `is-invalid` por 2s.

Registro no servidor

Ao confirmar, o front chama `POST /api/tcp/users/register` e passa a aparecer na lista de online.

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
Nome de usuário	Texto livre	Sim	Não pode ser vazio (cliente). No servidor também é rejeitado se vazio ou já em uso. Sem limite de tamanho; aceita espaços e acentos.

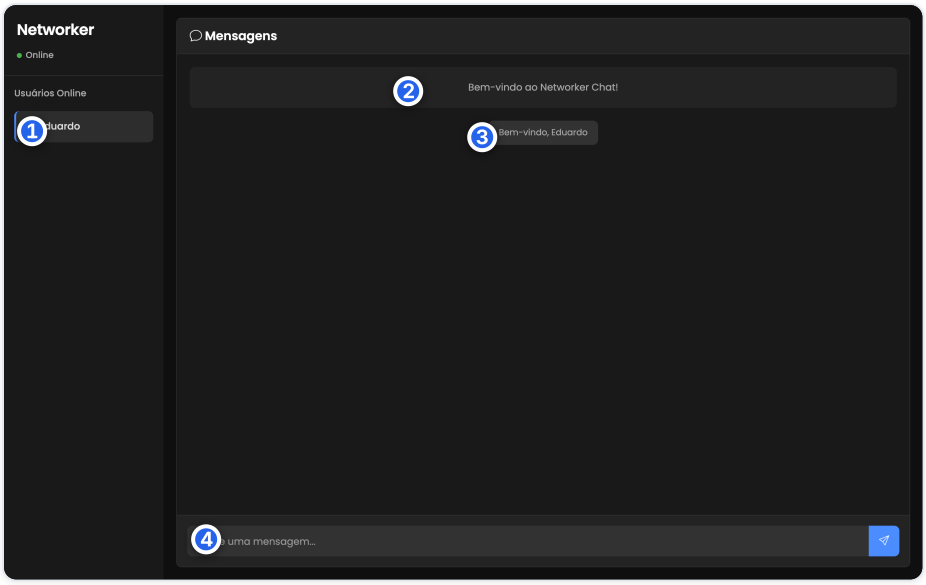
OBSERVAÇÃO

O backend recusa nome duplicado (`{"success":false,"message":"Nome de usuário já está em uso"}`), mas a interface web hoje **não trata essa resposta** e entra no chat mesmo assim. No cliente TCP a recusa é explícita: `ERRO: Nome de usuário já em uso` .

TELA

Área de chat — estado inicial

Depois de entrar, o usuário vê o layout principal: barra lateral de contatos à esquerda e a conversa à direita. Sem histórico — a conversa começa vazia, com um aviso de boas-vindas local.



1. O próprio usuário aparece destacado na lista (faixa azul à esquerda, estilo `user-self`).
2. Faixa fixa “Bem-vindo ao Networker Chat!”.
3. Mensagem de sistema “Bem-vindo, <nome>”, gerada apenas no navegador local.
4. Campo de mensagem com botão de envio; `Enter` também envia.

Sem histórico

Mensagens não são persistidas. Quem entra depois não vê nada do que foi dito antes.

Boas-vindas local

O “Bem-vindo, <nome>” não é enviado aos outros participantes — é só um retorno visual.

Canal único

Tudo que se digita aqui vai para todos: a UI web não expõe conversa privada.

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
Mensagem	Texto livre	Sim para enviar	Ignora envio se vazia após *trim*. `Enter` envia. `autocomplete="off"`. Limite prático de 8 KB imposto pelo servidor WebSocket.

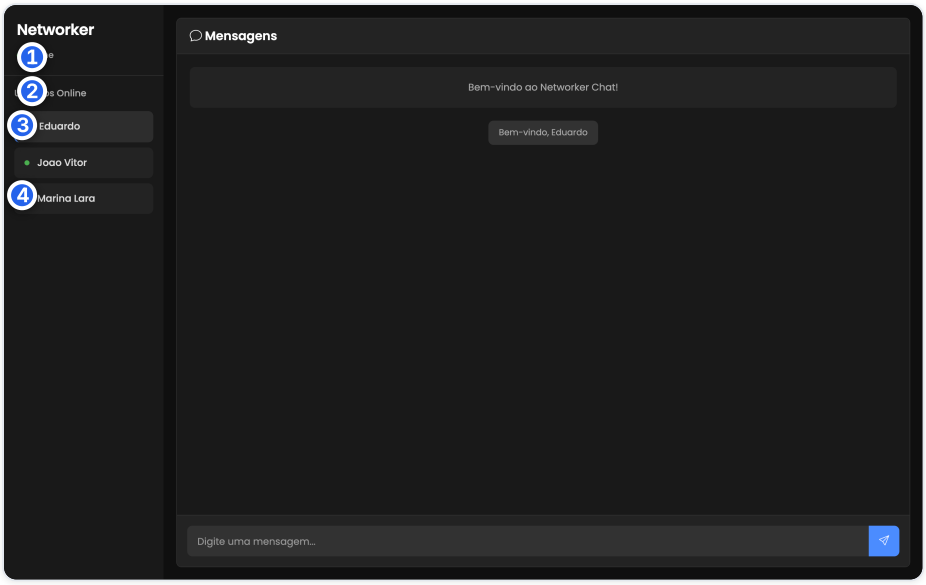
OBSERVAÇÃO

A tela é single-page: não há rotas internas nem recarregamento. Todos os “estados” deste manual são variações desta mesma tela conforme as mensagens e a lista de usuários mudam.

TELA

Usuários online

A barra lateral lista quem está conectado no momento — usuários da web e clientes TCP compartilham a mesma lista. Ela é atualizada por *polling* a cada 5 segundos e também por eventos WebSocket.



1. Indicador global “Online” com ponto verde no topo da barra lateral.
2. Título da seção “Usuários Online”.
3. O usuário atual sempre com a faixa azul de destaque.
4. Demais participantes, cada um com ponto de status verde.

Fonte única

Web e TCP entram no mesmo mapa `username → conexão` no servidor; a lista não distingue a origem.

Atualização

`GET /api/tcp/users` a cada 5s + *push* em `/topic/users` a cada entrada/saída.

Sem histórico de presença

Sair fecha a conexão e remove o nome silenciosamente; não há aviso de “fulano saiu”.

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
— (somente leitura)	Lista	—	Cada item mostra ponto de status + nome. Vazio exibe “Nenhum usuário conectado”; erro de carga exibe “Erro ao carregar usuários”.

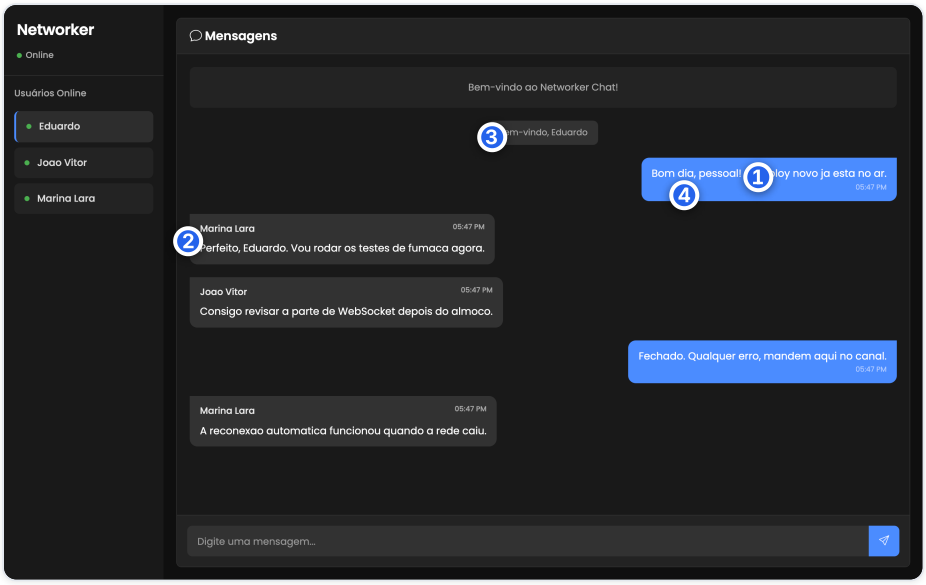
OBSERVAÇÃO

Não há papéis de acesso. Todos os participantes são iguais: não existe administrador, moderação, expulsão ou permissões — qualquer um pode enviar mensagens e broadcasts.

TELA

Conversa em grupo

Com vários participantes, o chat mostra os três tipos de bolha: mensagens próprias (à direita, azul), de outros (à esquerda, com nome e horário) e de sistema (centralizadas, discretas).



1. Mensagem própria: alinhada à direita, fundo azul, horário abaixo.
2. Mensagem de outro usuário: à esquerda, com cabeçalho de nome + horário.
3. Mensagem de sistema: centralizada, cinza, sem autor.
4. Horário no formato local `HH:MM` (derivado do `timestamp` do servidor).

Entrega

O texto vai para `/app/chat` e o servidor reencaminha a todos em `/topic/messages`.

Rolagem automática

A cada mensagem recebida a área rola para o fim (`scrollToBottom`).

Carimbo de tempo

O servidor grava `yyyy-MM-dd HH:mm:ss`; o cliente exibe só horas e minutos.

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
Mensagem	Texto livre	Sim	Enviada como tipo `BROADCAST` para `/app/chat`. Sem edição nem exclusão depois de enviada.

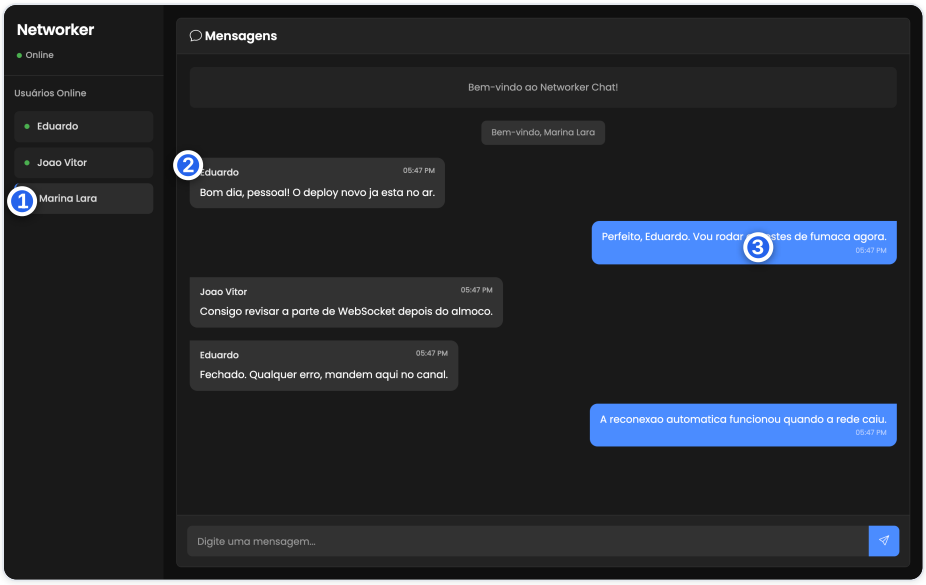
OBSERVAÇÃO

Todas as mensagens da UI web são públicas. Mensagens privadas existem apenas no protocolo TCP (`MSG:destinatário:texto`) ou via `POST /api/tcp/message`.

TELA

Conversa — visão de outro participante

A mesma conversa da página anterior, agora na aba de outra pessoa (Marina). O que muda é apenas a perspectiva: as bolhas de quem está olhando vão para a direita; as demais, para a esquerda.



1. Na lista lateral, agora Marina é quem aparece com a faixa azul de “eu”.
2. As mensagens de Eduardo aparecem à esquerda, com cabeçalho.
3. As mensagens da própria Marina vão para a direita, em azul.

Sem servidor de identidade

O “eu” é decidido no navegador comparando `message.sender` com o nome digitado.

Nomes repetidos confundem

Dois participantes com o mesmo nome seriam ambos tratados como “eu” na tela.

Consistência

Ordem e conteúdo das mensagens são idênticos entre abas; só o alinhamento muda.

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
— (somente leitura)	Conversa	—	Renderização por participante. Não há confirmação de leitura, digitação (“typing...”) ou entrega.

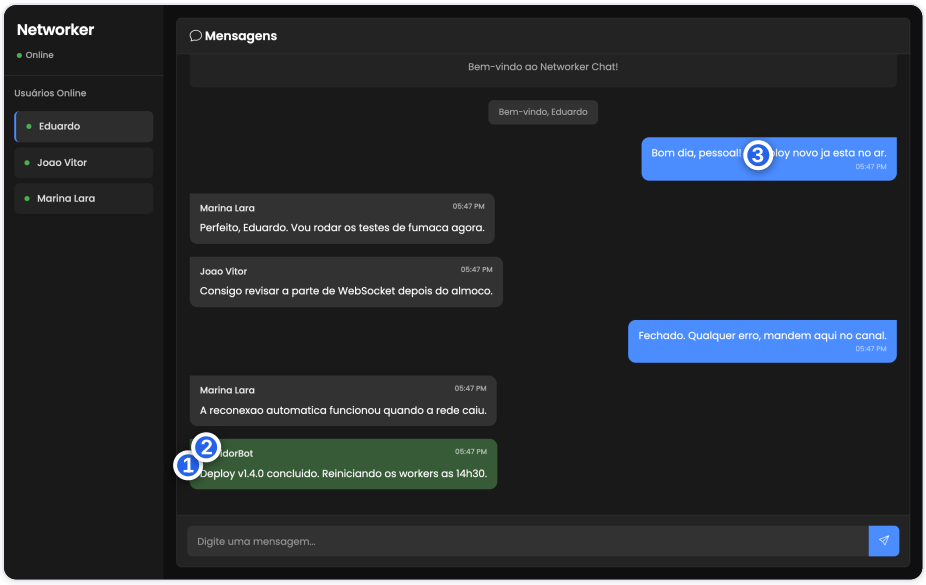
OBSERVAÇÃO

Abrir duas abas no mesmo navegador é a forma mais rápida de testar o chat: cada aba mantém a sua própria sessão e nome.

TELA

Mensagem de broadcast via TCP

Um cliente conectado ao servidor TCP (porta 12345) que envia `BROADCAST:<texto>` tem a mensagem entregue também aos usuários da web — com um estilo próprio, em verde.



1. Bolha de broadcast em verde, visualmente distinta das mensagens comuns.
2. Autor identificado (aqui `ServidorBot`), origem no cliente TCP.
3. Mensagens normais do chat seguem o estilo padrão azul/cinza.

Ponte TCP → Web

`TcpService.broadcastMessage` chama `notifyBroadcastMessage`, publicado em `/topic/broadcast-messages`.

Uso típico

Avisos operacionais de um processo/robô: deploy concluído, manutenção, reinício de serviço.

Pré-requisito

O servidor TCP precisa estar ligado — `POST /api/tcp/server/start` (inicia parado por padrão).

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
BROADCAST:<texto>	Comando TCP	Sim (prefixo)	Tudo após `BROADCAST:` é a mensagem. Entregue como `BROADCAST <remetente>: <texto>` aos clientes TCP e em bolha verde na web.

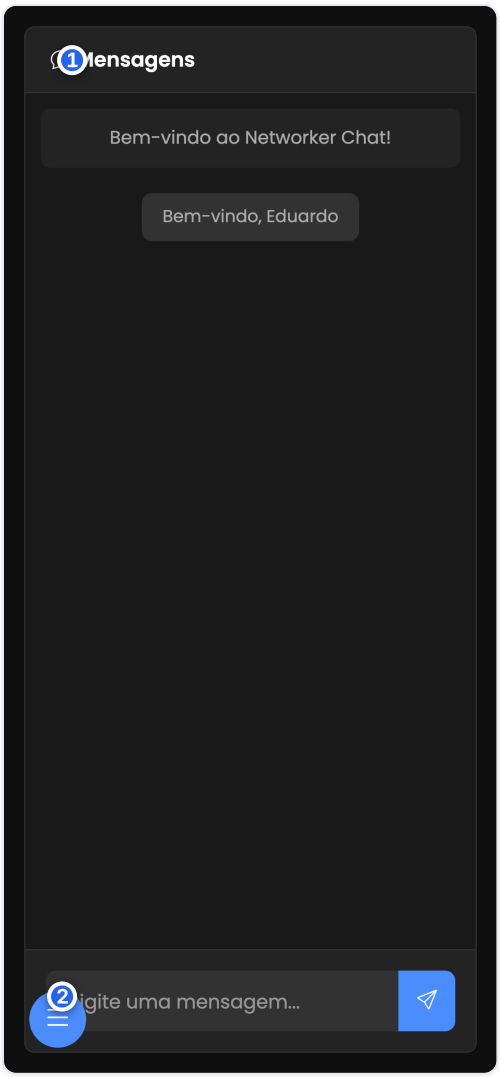
OBSERVAÇÃO

A UI web não tem botão de “broadcast”: mensagens digitadas no navegador usam `/topic/messages` (estilo comum). O verde só aparece quando a origem é o protocolo TCP.

TELA

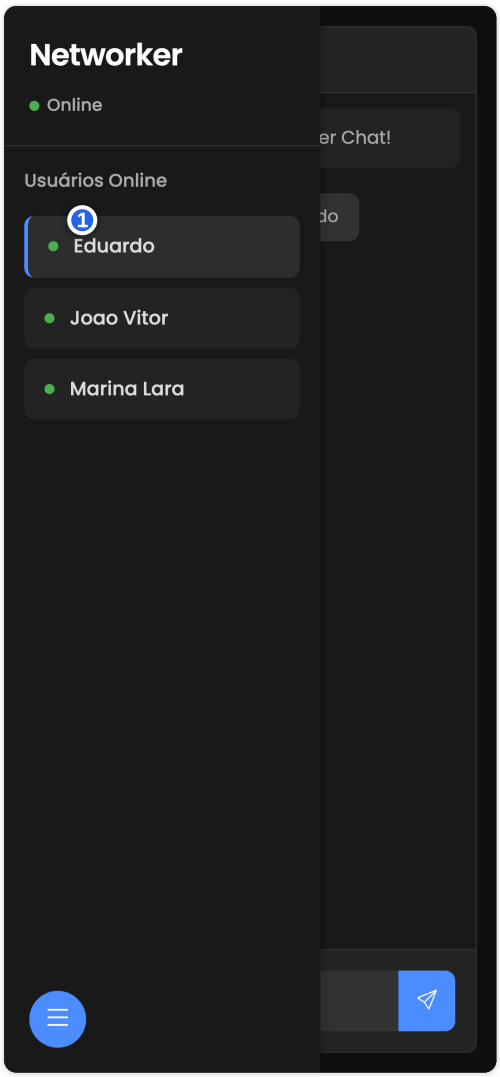
Layout responsivo (mobile)

Abaixo de 992 px a barra lateral sai do fluxo e vira um painel deslizante, acionado por um botão flutuante no canto inferior esquerdo.



1 Cabeçalho “Mensagens” ocupa toda a largura; a conversa fica em tela cheia.

2 Botão flutuante que abre/fecha a lista de usuários.



1 Painel de usuários deslizando sobre o chat (segunda imagem).

Breakpoints
≤ 992 px: barra lateral vira *drawer*. ≤ 576 px: espaçamentos e fontes do chat diminuem.

Interação
O botão alterna a classe `show` na `.sidebar`; toque fora não fecha o painel.

Pontos de atenção
O painel não escurece o fundo e o botão flutuante se sobrepõe ao campo de mensagem.

CAMPOS E PARÂMETROS

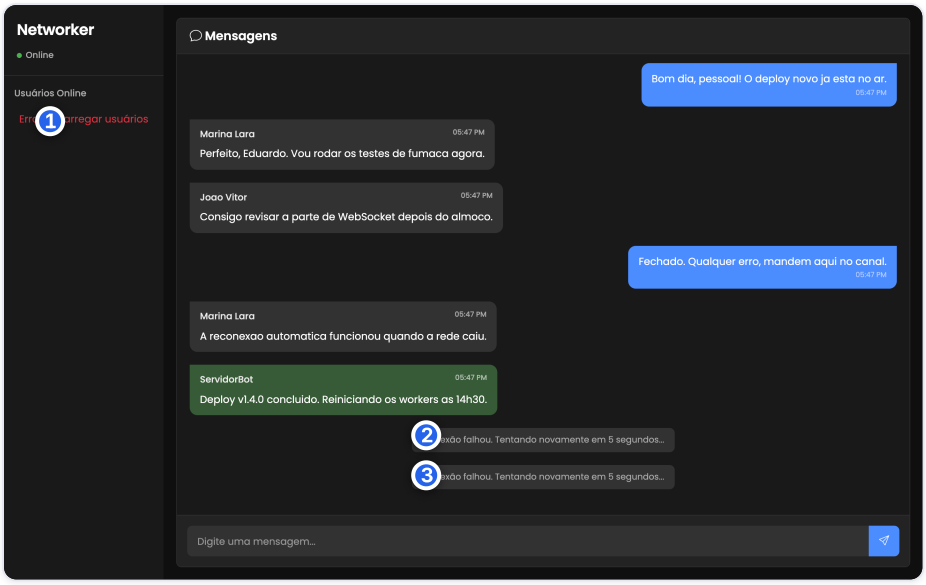
CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
— (somente leitura)	Layout	—	Mesma tela e mesmos campos da versão desktop; muda apenas a disposição. Testado em viewport de 390×844.

OBSERVAÇÃO
Como não há backdrop, em telas estreitas convém fechar o painel manualmente no mesmo botão antes de digitar.

⚠ TELA

Conexão perdida e reconexão

Se o servidor cai ou a rede falha, o cliente detecta a queda do WebSocket, avisa na conversa e tenta reconectar sozinho a cada 5 segundos. O *polling* da lista de usuários passa a mostrar erro.



1. “Erro ao carregar usuários” (vermelho) quando `GET /api/tcp/users` falha.
2. Mensagem de sistema “Conexão falhou. Tentando novamente em 5 segundos...”.
3. A tentativa se repete a cada 5 s; cada falha acrescenta uma nova linha.

Detecção

Callback de erro do `stompClient.connect` dispara o aviso e reagenda `connectWebSocket`.

Recuperação

Quando o servidor volta, a próxima tentativa reconecta e a lista de usuários se refaz.

Mensagens locais

O que estava na tela permanece; nada é reenviado nem recuperado do servidor.

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
— (somente leitura)	Estado	—	Disparado por queda de `/ws` ou de `/api/tcp/*`. Sem limite de tentativas e sem botão de “reconectar agora”.

⚠ **OBSERVAÇÃO**

Ao sair da página o app mostra a confirmação nativa do navegador (“Você será desconectado se sair da página.”) e envia um *beacon* de `unregister`.

TELA

Integração — API REST

Além da tela, o serviço expõe uma API sob `/api/tcp` para operar o servidor TCP, consultar usuários e diagnosticar. Respostas em JSON; CORS liberado para qualquer origem.

GET /api/tcp/server/status

GET /api/tcp/debug/users

MÉTODO	ROTA	DESCRIÇÃO
GET	`/api/tcp/server/status`	Estado do servidor TCP: `{running, port, message}`.
POST	`/api/tcp/server/start`	Liga o servidor TCP (aceita `?port=`).
POST	`/api/tcp/server/stop`	Desliga o servidor e fecha todas as conexões.
GET	`/api/tcp/users`	Lista de nomes conectados (web + TCP).
GET	`/api/tcp/users/{nome}/status`	`User is online` / `User is offline`.
GET	`/api/tcp/debug/users`	Diagnóstico: usuários, contagem, status e porta.
POST	`/api/tcp/users/register`	Registra usuário web (corpo `{"username"}`).
POST	`/api/tcp/message`	Mensagem direta `sender→recipient` (offline ⇒ `success:false`).

Ciclo do servidor TCP

`server/start` · `server/stop` · `server/status`. Começa parado na porta 12345.

Usuários

`GET users`, `GET users/{nome}/status`, `POST users/register` e `users/unregister`.

Diagnóstico

`GET /api/tcp/debug/users` devolve usuários, contagem, status e porta em um único objeto.

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
serverAddress, port, message	Query params	Sim	`POST /api/tcp/send` — abre um socket, envia a linha e devolve a resposta do servidor.
sender, recipient, message	Query params	Sim	`POST /api/tcp/message` — entrega direta a um usuário; se offline, `success:false`.

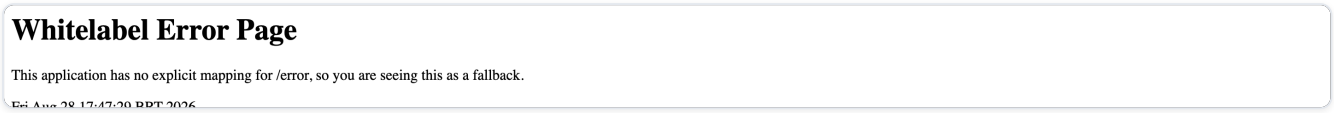
OBSERVAÇÃO

A API não tem autenticação. Com `allowCredentials(true)` e `allowedOriginPatterns("*")`, qualquer página pode chamá-la — aceitável em laboratório, não em produção.

TELA

Comportamento da rota raiz

Um detalhe importante para quem for publicar: acessar `http://localhost:8080/` retorna HTTP 500. A aplicação real é servida em `/index.html`.



GET http://localhost:8080/ — resposta 500

Causa

`HomeController` retorna a *view* `"index"`, mas `@EnableWebMvc` remove o resolvedor padrão e o Thymeleaf está desativado.

Efeito

Resposta: `{"status":500,"error":"Internal Server Error","message":"Could not resolve view with name 'index'"}`.

Contorno

Usar `/index.html` (servido pelo *resource handler*) ou publicar um redirecionamento `/ → /index.html`.

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
/	Rota GET	—	Retorna 500 no estado atual. Documente o acesso pela URL `/index.html` para os usuários finais.

OBSERVAÇÃO

Nenhuma outra rota é afetada: `/index.html`, os arquivos estáticos e toda a API `/api/tcp/*` respondem normalmente.

TELA

Protocolo TCP e cliente de terminal

O núcleo do Networker é um servidor TCP textual, orientado a linha, na porta 12345. Abaixo, uma sessão real com dois clientes (`ana` e `bruno`) exercitando todos os comandos.

sessão TCP · porta 12345

```
$ nc localhost 12345        # cliente A (ana)
  ← SUCESSO: Logado como ana
  ← USUARIOS_ONLINE: ana

$ nc localhost 12345        # cliente B (bruno)
  ← SUCESSO: Logado como bruno
  ← USUARIOS_ONLINE: ana bruno

  → [ana]  LISTAR_USUARIOS
  ← USUARIOS_ONLINE: ana bruno

  → [bruno] MSG:ana:Sobe o ambiente de homologacao, por favor
  ← [ana recebe] DE bruno: Sobe o ambiente de homologacao, por favor

  → [ana]  BROADCAST:Homologacao no ar as 15h
  ← [bruno recebe] BROADCAST ana: Homologacao no ar as 15h

  → [bruno] COMANDO_ERRADO
  ← ERRO: Comando inválido. Comandos válidos: MSG:destinatario:mensagem, SAIR, LISTAR_USUARIOS, BROADCAST:mensagem

  → [ana] SAIR    → [bruno] SAIR
  conexoes encerradas
```

Saída real capturada do servidor durante a execução local.

LOGIN obrigatório

A 1ª linha precisa ser `LOGIN:<nome>`. Caso contrário: `ERRO: Login necessário` e a conexão cai.

MSG e BROADCAST

`MSG:destino:texto` entrega `DE <remetente>: …`; `BROADCAST:texto` vai a todos.

LISTAR_USUARIOS / SAIR

Consulta a lista (`USUARIOS_ONLINE: …`) e encerra a sessão de forma limpa.

CAMPOS E PARÂMETROS

CAMPO	TIPO	OBRIGATÓRIO	REGRAS E VALIDAÇÕES
LOGIN:<nome>	Comando	Sim (1ª linha)	Nome vazio ou repetido é recusado. Em sucesso: `SUCESSO: Logado como <nome>` + lista de online.
MSG:<destino>:<texto>	Comando	Sim	Destino offline → `ERRO: Não foi possível enviar mensagem para <destino>`.
BROADCAST:<texto>	Comando	Sim	Entrega a todos os clientes TCP e à web (bolha verde).
LISTAR_USUARIOS · SAIR	Comando	Sim	Comando desconhecido devolve `ERRO: Comando inválido…` com a lista de comandos válidos.

OBSERVAÇÃO

A classe `TcpClient` do repositório é um cliente de referência pronto: pede o nome, faz `LOGIN` e repassa o que você digitar. Feito para rodar em terminal, não é parte da interface web.

REGRAS

Regras de negócio e decisões de UX

Consolidação do que foi observado percorrendo o sistema em execução. Onde o comportamento atual é incompleto, está marcado como tal.

TEMA	COMPORTAMENTO OBSERVADO
Identificação	Nome de exibição obrigatório; sem senha, cadastro ou recuperação. Sessão vive só na aba (fechou, desconectou).
Unicidade de nome	Servidor recusa nome repetido (TCP e REST). A UI web ainda não trata essa recusa e entra assim mesmo.
Papéis de acesso	Inexistentes. Todos os participantes têm exatamente os mesmos poderes; não há moderação nem administração.
Visibilidade das mensagens	Chat da web é sempre público (para todos). Mensagem privada só via TCP (`MSG:`) ou `POST /api/tcp/message` .
Histórico	Nenhum. Sem banco de dados; nada é persistido; quem chega depois não vê o passado.
Servidor TCP	Inicia desligado. Sobe/desce por `POST /api/tcp/server/start` e `/server/stop` ; porta padrão 12345, configurável.
Reconexão	Automática a cada 5 s quando o WebSocket cai; a lista de usuários exibe erro enquanto o backend não volta.
Limites	Mensagem WebSocket até 8 KB (texto e binário). Sem limite de tamanho para o nome de usuário.
CORS	Liberado para qualquer origem, com credenciais; métodos GET/POST/PUT/DELETE/OPTIONS.
Rota raiz	`GET /` responde 500 no estado atual; a aplicação é acessada por `/index.html` .

⚠ **OBSERVAÇÃO**

Segurança: a aplicação não tem autenticação, autorização nem *rate limiting*, e o CORS está aberto. É adequada para laboratório e demonstração; para expor em rede, seria necessário adicionar identidade e limitar origens.

REFERÊNCIA

Glossário e canais

TERMO	SIGNIFICADO NO NETWORKER
STOMP	Protocolo de mensagens sobre WebSocket usado entre o navegador e o Spring. Canais em `/topic/*`.
SockJS	Camada de compatibilidade que emula WebSocket quando ele não está disponível. Endpoint `/ws`.
Broadcast	Mensagem enviada a todos os conectados. Na web tem estilo verde quando vem do protocolo TCP.
Cliente TCP	Qualquer programa que fale o protocolo textual na porta 12345 (ex.: a classe `TcpClient`, `nc`, `telnet`).
Usuário WebSocket	Participante vindo do navegador. É guardado no mesmo mapa dos clientes TCP, porém sem socket associado.
/topic/messages	Canal do chat público. `/topic/users` carrega a lista de online; `/topic/broadcast-messages`, os avisos.

COMANDOS DO PROTOCOLO TCP

COMANDO	RESPOSTA / EFEITO
`LOGIN:<nome>`	`SUCESSO: Logado como <nome>` + `USUARIOS_ONLINE: …`
`MSG:<destino>:<texto>`	Destino recebe `DE <remetente>: <texto>`
`BROADCAST:<texto>`	Todos recebem `BROADCAST <remetente>: <texto>`
`LISTAR_USUARIOS`	`USUARIOS_ONLINE: a b c`
`SAIR`	Encerra a conexão
(outro)	`ERRO: Comando inválido…`

ENCERRAMENTO

Créditos e escopo

Este manual cobre o repositório `networker` (chat Spring Boot). O repositório `academix`, também do mesmo autor, é uma aplicação **desktop Java Swing** de gestão escolar, com telas construídas no editor de formulários da IDE e marcada como não finalizada — fora do escopo de um catálogo baseado em capturas de navegador.

COMO ESTE DOCUMENTO FOI PRODUZIDO

1. Clonagem dos repositórios em pasta isolada e leitura do código-fonte.
2. Execução local do `networker` (`mvn spring-boot:run`, porta 8080).
3. Sessões reais de chat com três participantes e um cliente TCP, dirigidas por navegador headless.
4. Capturas em alta resolução de cada estado (1440×900 no desktop, 390×844 no mobile).
5. Anotação das regras de negócio e decisões de UX visíveis em tela e no código.
6. Encerramento dos processos locais subidos para a produção do manual.

`github.com/eduardofabrii/networker`

Stack: Java · Spring Boot 2.7.16 · WebSocket/STOMP · SockJS · Bootstrap 5 · sockets TCP

Documento de referência — Versão 1.0 — Agosto de 2026