<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width,initial-scale=1" />

<title>Real-time Chat (Single File)</title>

<style>

:root{--bg:#0f1724;--card:#0b1220;--accent:#06b6d4;--muted:#94a3b8;--me:#083344}

\*{box-sizing:border-box;font-family:Inter,ui-sans-serif,system-ui,Segoe UI,Roboto,"Helvetica Neue",Arial}

html,body{height:100%;margin:0;background:linear-gradient(180deg,#071023 0%, #071923 100%);color:#e6eef6}

.app{max-width:980px;margin:28px auto;padding:20px;border-radius:12px;background:linear-gradient(180deg, rgba(255,255,255,0.02), rgba(255,255,255,0.01));box-shadow:0 8px 30px rgba(2,6,23,0.7)}

header{display:flex;align-items:center;justify-content:space-between;margin-bottom:16px}

header h1{font-size:18px;margin:0}

.controls{display:flex;gap:8px;align-items:center}

input[type=text].name{padding:8px 10px;border-radius:8px;border:1px solid rgba(255,255,255,0.04);background:transparent;color:inherit;width:180px}

.status{font-size:12px;color:var(--muted)}.layout{display:grid;grid-template-columns:1fr 320px;gap:14px}

.chat{background:var(--card);padding:14px;border-radius:10px;min-height:420px;display:flex;flex-direction:column}

.messages{flex:1;overflow:auto;padding-right:6px}

.msg{display:flex;gap:10px;margin-bottom:12px;align-items:flex-end}

.avatar{width:36px;height:36px;border-radius:10px;background:#071a26;color:var(--accent);display:flex;align-items:center;justify-content:center;font-weight:600}

.bubble{max-width:68%;padding:10px 12px;border-radius:10px;background:rgba(255,255,255,0.03);font-size:14px}

.meta{font-size:11px;color:var(--muted);margin-top:6px}

.me{margin-left:auto;flex-direction:row-reverse}

.me .bubble{background:linear-gradient(180deg, rgba(6,182,212,0.12), rgba(6,182,212,0.08));color:var(--accent);border-radius:10px 10px 6px 10px}

.composer{display:flex;gap:8px;margin-top:12px}

textarea{flex:1;min-height:44px;max-height:160px;padding:10px;border-radius:10px;border:1px solid rgba(255,255,255,0.03);resize:none;background:transparent;color:inherit}

button{background:var(--accent);border:none;padding:10px 14px;border-radius:10px;color:#002;cursor:pointer;font-weight:600}

aside{background:linear-gradient(180deg, rgba(255,255,255,0.02), rgba(255,255,255,0.01));padding:12px;border-radius:10px;height:420px}

.info h3{margin:0 0 8px 0;font-size:14px}

.peers{font-size:13px;color:var(--muted);margin-top:8px}

.typing{font-size:13px;color:var(--muted);height:20px}

.footer-note{font-size:12px;color:var(--muted);margin-top:12px}

@media (max-width:880px){.layout{grid-template-columns:1fr}.app{margin:12px}}

</style>

</head>

<body>

<div class="app" role="application" aria-label="Real-time chat single file">

<header>

<h1>Real-time Chat (single HTML)</h1>

<div class="controls">

<input class="name" id="username" type="text" placeholder="Your name (eg: Anand)" maxlength="24" />

<div class="status" id="connStatus">Disconnected</div>

</div>

</header><div class="layout">

<section class="chat" aria-live="polite">

<div class="messages" id="messages" tabindex="0"></div>

<div class="composer">

<textarea id="text" placeholder="Type a message and press Send — supports Enter for send, Shift+Enter for newline"></textarea>

<button id="sendBtn">Send</button>

</div>

<div class="typing" id="typing"></div>

</section>

<aside>

<div class="info">

<h3>Connection</h3>

<p style="margin:0">This demo will try a WebSocket connection to <code id="wsUrl">ws://localhost:8080</code>. If not available, it falls back to BroadcastChannel for real-time between browser tabs (local only).</p>

<div class="peers" id="peers">Peers: 1 (you)</div>

<div class="footer-note">Messages persist locally (browser). Open another tab to test BroadcastChannel fallback.</div>

</div>

</aside>

</div>

</div> <script>

// Configuration

const WS\_URL = 'ws://localhost:8080'; // change to your websocket server if you have one

const STORAGE\_KEY = 'simple\_chat\_messages\_v1';

// Elements

const usernameEl = document.getElementById('username');

const messagesEl = document.getElementById('messages');

const textEl = document.getElementById('text');

const sendBtn = document.getElementById('sendBtn');

const connStatusEl = document.getElementById('connStatus');

const typingEl = document.getElementById('typing');

const peersEl = document.getElementById('peers');

const wsUrlEl = document.getElementById('wsUrl');

wsUrlEl.textContent = WS\_URL;

// State

let username = localStorage.getItem('chat\_username') || '';

usernameEl.value = username;

let socket = null;

let bc = null; // BroadcastChannel fallback

let peers = new Set();

let typingTimeout = null;

// Util

const nowISO = ()=> new Date().toISOString();

const formatTime = t=> new Date(t).toLocaleTimeString();

// Load persisted messages

let messages = JSON.parse(localStorage.getItem(STORAGE\_KEY) || '[]');

function renderMessages(){

messagesEl.innerHTML = '';

for(const m of messages){

const el = renderMessage(m);

messagesEl.appendChild(el);

}

messagesEl.scrollTop = messagesEl.scrollHeight;

}

function renderMessage(m){

const c = document.createElement('div');

c.className = 'msg ' + (m.from === username ? 'me' : '');

const avatar = document.createElement('div');

avatar.className='avatar';

avatar.textContent = (m.from || 'Anon').slice(0,2).toUpperCase();

const bubble = document.createElement('div');

bubble.className='bubble';

bubble.innerHTML = '<strong>'+escapeHtml(m.from||'Anon')+'</strong><div style="margin-top:6px">'+escapeHtml(m.text)+'</div><div class="meta">'+formatTime(m.ts)+'</div>';

c.appendChild(avatar);

c.appendChild(bubble);

return c;

}

function pushMessage(m, persist=true){

messages.push(m);

if(persist) localStorage.setItem(STORAGE\_KEY, JSON.stringify(messages));

messagesEl.appendChild(renderMessage(m));

messagesEl.scrollTop = messagesEl.scrollHeight;

}

function escapeHtml(s){return String(s).replace(/&/g,'&amp;').replace(/</g,'&lt;').replace(/>/g,'&gt;')}

// Send via active channel (WebSocket preferred, else BroadcastChannel)

function sendPacket(obj){

const payload = JSON.stringify(obj);

if(socket && socket.readyState === WebSocket.OPEN){

socket.send(payload);

} else if(bc){

bc.postMessage(obj);

} else {

// Local echo

handleIncoming(obj, true);

}

}

// Handle incoming packets

function handleIncoming(obj, self=false){

if(obj.type === 'msg'){

pushMessage({from:obj.from,text:obj.text,ts:obj.ts}, !self);

} else if(obj.type === 'presence'){

peers.add(obj.from);

peersEl.textContent = 'Peers: ' + peers.size;

} else if(obj.type === 'typing'){

typingEl.textContent = obj.from + ' is typing...';

clearTimeout(typingTimeout);

typingTimeout = setTimeout(()=> typingEl.textContent = '', 1200);

}

}

// WebSocket connection attempt

function connectWebSocket(){

try{

socket = new WebSocket(WS\_URL);

connStatusEl.textContent = 'Connecting...';

socket.addEventListener('open', ()=>{

connStatusEl.textContent = 'Connected (WebSocket)';

sendPacket({type:'presence',from:username,ts:nowISO()});

});

socket.addEventListener('message', ev=>{

try{const obj = JSON.parse(ev.data); handleIncoming(obj);}catch(e){console.warn('bad ws msg',e)}

});

socket.addEventListener('close', ()=>{

connStatusEl.textContent = 'WebSocket closed — using local BroadcastChannel';

socket = null; startBroadcastChannel();

});

socket.addEventListener('error', (e)=>{

connStatusEl.textContent = 'WebSocket error — using local BroadcastChannel';

socket.close(); socket = null; startBroadcastChannel();

});

}catch(e){

connStatusEl.textContent = 'WebSocket failed — using local BroadcastChannel';

socket = null; startBroadcastChannel();

}

}

// BroadcastChannel fallback (works between tabs on same origin)

function startBroadcastChannel(){

if(bc) return;

try{

bc = new BroadcastChannel('simple\_realtime\_chat\_channel');

bc.onmessage = (ev)=> handleIncoming(ev.data);

connStatusEl.textContent = 'Connected (BroadcastChannel)';

// announce presence

bc.postMessage({type:'presence',from:username,ts:nowISO()});

}catch(e){

connStatusEl.textContent = 'Offline (no realtime channel)';

}

}

// UI wiring

sendBtn.addEventListener('click', ()=> doSend());

textEl.addEventListener('keydown', (e)=>{

if(e.key === 'Enter' && !e.shiftKey){ e.preventDefault(); doSend(); }

else{ // typing

sendPacket({type:'typing',from:username,ts:nowISO()});

}

});

usernameEl.addEventListener('change', ()=>{

username = usernameEl.value.trim() || 'Anon';

localStorage.setItem('chat\_username', username);

sendPacket({type:'presence',from:username,ts:nowISO()});

});

function doSend(){

const text = textEl.value.trim();

if(!text) return;

const packet = {type:'msg',from:username || 'Anon',text,ts:nowISO()};

// local echo

pushMessage({from:packet.from,text:packet.text,ts:packet.ts}, true);

sendPacket(packet);

textEl.value = '';

textEl.focus();

}

// Initial setup

(function init(){

if(!username){ username = 'Anon' + Math.floor(Math.random()\*999); usernameEl.value = username; localStorage.setItem('chat\_username', username); }

renderMessages();

connectWebSocket();

// if websocket doesn't exist after 700ms, ensure BroadcastChannel started

setTimeout(()=>{ if(!socket) startBroadcastChannel(); }, 700);

// Keep peer count minimal: use presence message aggregator via localStorage (simple)

window.addEventListener('storage', (e)=>{

if(e.key === STORAGE\_KEY) renderMessages();

});

// Make messages accessible via keyboard

messagesEl.addEventListener('keydown', (e)=>{ if(e.key==='Home') messagesEl.scrollTop = 0; });

})();

// Small safety: try to gracefully close channels on unload

window.addEventListener('beforeunload', ()=>{

try{ if(socket) socket.close(); if(bc) bc.close(); }catch(e){}

});

</script></body>

</html>