<!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>

/\* Simple modern styling \*/

:root{--bg:#0f1724;--card:#0b1220;--muted:#94a3b8;--accent:#06b6d4;--me:#0ea5a4}

\*{box-sizing:border-box}

html,body{height:100%;margin:0;font-family:Inter,ui-sans-serif,system-ui,Segoe UI,Roboto,'Helvetica Neue',Arial}

body{background:linear-gradient(180deg,#071124 0%, #071829 100%);color:#e6eef6;display:flex;align-items:center;justify-content:center;padding:18px}

.app{width:100%;max-width:980px;height:85vh;background:linear-gradient(180deg,rgba(255,255,255,0.02),rgba(255,255,255,0.01));border-radius:12px;overflow:hidden;display:grid;grid-template-columns:320px 1fr}/\* sidebar \*/

.sidebar{padding:14px;border-right:1px solid rgba(255,255,255,0.03);display:flex;flex-direction:column;gap:12px}

.brand{display:flex;gap:10px;align-items:center}

.logo{width:44px;height:44px;border-radius:10px;background:linear-gradient(135deg,var(--accent),#7c3aed);display:flex;align-items:center;justify-content:center;font-weight:700}

.title{font-weight:600}

.mebox{display:flex;gap:10px;align-items:center;padding:8px;border-radius:8px;background:rgba(255,255,255,0.02)}

.mebox input{background:transparent;border:0;color:inherit;outline:none;font-weight:600}

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

.control input,.control button{width:100%;padding:8px;border-radius:8px;border:1px solid rgba(255,255,255,0.04);background:transparent;color:var(--muted);outline:none}

/\* chat area \*/

.chat{display:flex;flex-direction:column;height:100%}

.messages{flex:1;overflow:auto;padding:20px;display:flex;flex-direction:column;gap:12px}

.day{color:var(--muted);text-align:center;font-size:12px}

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

.avatar{width:44px;height:44px;border-radius:10px;background:linear-gradient(135deg,#334155,#0ea5a4);display:flex;align-items:center;justify-content:center;font-weight:700}

.bubble{max-width:70%;padding:10px 12px;border-radius:10px;background:rgba(255,255,255,0.03);line-height:1.25}

.meta{font-size:12px;color:var(--muted);margin-top:6px}

.right{flex-direction:row-reverse}

.right .bubble{background:linear-gradient(135deg,var(--me),#34d399);color:#012;}

.composer{padding:12px;border-top:1px solid rgba(255,255,255,0.03);display:flex;gap:8px;align-items:center}

.composer textarea{flex:1;resize:none;padding:10px;border-radius:8px;border:1px solid rgba(255,255,255,0.03);background:transparent;color:inherit;min-height:46px}

.btn{padding:10px 12px;border-radius:8px;border:0;background:var(--accent);color:#012;font-weight:600;cursor:pointer}

.tiny{font-size:12px;color:var(--muted)}

.typing{font-size:13px;color:var(--muted);padding-left:8px}

.attachment-preview{max-width:240px;border-radius:8px;display:block;margin-top:8px}

@media (max-width:820px){.app{grid-template-columns:1fr}.sidebar{display:none}}

</style>

</head>

<body>

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

<aside class="sidebar">

<div class="brand"><div class="logo">RT</div><div><div class="title">Realtime Chat</div><div class="tiny">Single-file demo — works across tabs</div></div></div><div class="mebox" title="Edit your display name">

<div class="avatar" id="meAvatar">Y</div>

<input id="username" placeholder="Your name" maxlength="32" />

</div>

<div class="control">

<input id="channel" placeholder="Channel (room)" value="global" />

</div>

<div style="display:flex;gap:8px;align-items:center">

<input id="wsUrl" placeholder="Optional WebSocket URL (ws://...)" />

<button id="wsConnect" class="btn" style="padding:8px">Connect</button>

</div>

<div class="tiny">Tip: open this file in multiple browser tabs/windows to see 'real-time' via localStorage events. Or connect to a WebSocket server for networked realtime.</div>

<div style="margin-top:auto;font-size:12px;color:var(--muted)">Status: <span id="status">disconnected</span></div>

</aside>

<main class="chat">

<div style="padding:14px;border-bottom:1px solid rgba(255,255,255,0.02);display:flex;justify-content:space-between;align-items:center">

<div style="font-weight:600">Channel: <span id="channelLabel">global</span></div>

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

</div>

<div class="messages" id="messages" aria-live="polite"></div>

<div class="composer">

<input type="file" id="file" accept="image/\*" />

<textarea id="input" placeholder="Type a message — Enter to send, Shift+Enter for newline"></textarea>

<button id="send" class="btn">Send</button>

</div>

</main>

</div> <script>

// Single-file realtime chat using localStorage + optional WebSocket.

// Behavior: messages propagate across tabs via storage events. Each message stored under key rtc\_chat\_msg\_<channel>\_<id>

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

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

const channelInput = document.getElementById('channel');

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

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

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

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

const fileInput = document.getElementById('file');

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

const statusEl = document.getElementById('status');

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

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

const CLIENT\_ID = crypto.randomUUID();

let WS = null;

// load saved display name

usernameInput.value = localStorage.getItem('rtc\_chat\_name') || ('User-' + CLIENT\_ID.slice(0,4));

channelInput.value = localStorage.getItem('rtc\_chat\_channel') || 'global';

channelLabel.textContent = channelInput.value;

usernameInput.addEventListener('input', ()=>{

localStorage.setItem('rtc\_chat\_name', usernameInput.value);

meAvatar.textContent = initials(usernameInput.value);

});

meAvatar.textContent = initials(usernameInput.value);

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

localStorage.setItem('rtc\_chat\_channel', channelInput.value);

channelLabel.textContent = channelInput.value;

clearMessages();

loadHistory();

});

function initials(name){

if(!name) return 'U';

return name.split(' ').filter(Boolean).slice(0,2).map(n=>n[0].toUpperCase()).join('');

}

// send message function

async function sendMessage({text='',image=null}){

const channel = channelInput.value || 'global';

const msg = {

id: crypto.randomUUID(),

channel,

from: usernameInput.value || 'Anonymous',

fromId: CLIENT\_ID,

text,

image, // data URL if present

ts: Date.now()

};

// store as its own key so storage events fire reliably

const key = rtc\_chat\_msg\_${channel}\_${msg.id};

localStorage.setItem(key, JSON.stringify(msg));

// also bump a last pointer

localStorage.setItem(rtc\_chat\_last\_${channel}, msg.id);

// if websocket connected, send a compact payload

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

WS.send(JSON.stringify({type:'message',payload:msg}));

}

// locally append

appendMessage(msg, true);

// cleanup file input

fileInput.value = '';

}

// listen for storage events from other tabs

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

try{

if(!e.key) return;

if(e.key.startsWith('rtc\_chat\_msg\_')){

const raw = e.newValue;

if(!raw) return; // removed

const msg = JSON.parse(raw);

if(msg.fromId === CLIENT\_ID) return; // ignore our own duplicates

if(msg.channel !== (channelInput.value || 'global')) return; // other channel

appendMessage(msg, false);

}

if(e.key.startsWith('rtc\_chat\_typing\_')){

const raw = e.newValue && JSON.parse(e.newValue);

showTyping(raw);

}

if(e.key === rtc\_chat\_clear\_${channelInput.value}){

clearMessages();

}

}catch(err){console.warn('storage event parse',err)}

});

function appendMessage(msg, mine){

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

wrap.className = 'msg ' + (msg.fromId === CLIENT\_ID ? 'right' : '');

const avatar = document.createElement('div'); avatar.className='avatar'; avatar.textContent = initials(msg.from);

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

bubbleWrap.style.display='flex';

bubbleWrap.style.flexDirection='column';

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

if(msg.image){

const img = document.createElement('img'); img.src = msg.image; img.className='attachment-preview';

bubble.appendChild(img);

}

if(msg.text){

const p = document.createElement('div'); p.innerText = msg.text; bubble.appendChild(p);

}

const meta = document.createElement('div'); meta.className='meta';

const time = new Date(msg.ts);

meta.textContent = ${msg.from} · ${time.toLocaleTimeString()};

bubbleWrap.appendChild(bubble); bubbleWrap.appendChild(meta);

wrap.appendChild(avatar); wrap.appendChild(bubbleWrap);

messagesEl.appendChild(wrap);

messagesEl.scrollTop = messagesEl.scrollHeight - messagesEl.clientHeight + 100;

}

// load recent history by scanning localStorage keys (lightweight)

function loadHistory(){

const channel = channelInput.value || 'global';

const items = [];

for(let i=0;i<localStorage.length;i++){

const key = localStorage.key(i);

if(!key) continue;

if(key.startsWith(rtc\_chat\_msg\_${channel}\_)){

try{items.push(JSON.parse(localStorage.getItem(key)))}catch(e){}

}

}

items.sort((a,b)=>a.ts-b.ts);

items.forEach(m=>appendMessage(m,false));

}

function clearMessages(){ messagesEl.innerHTML=''; }

// typing indicator: write small ephemeral key

let typingTimer = null;

input.addEventListener('input', ()=>{

const channel = channelInput.value || 'global';

const payload = {name: usernameInput.value||'Anonymous',ts: Date.now(),id:CLIENT\_ID};

try{ localStorage.setItem(rtc\_chat\_typing\_${channel}, JSON.stringify(payload)); }catch(e){}

if(typingTimer) clearTimeout(typingTimer);

typingTimer = setTimeout(()=>{

try{ localStorage.removeItem(rtc\_chat\_typing\_${channel}); }catch(e){}

}, 2500);

});

function showTyping(raw){

const channel = channelInput.value || 'global';

if(!raw || raw.id === CLIENT\_ID || raw.name === usernameInput.value) { typingIndicator.textContent=''; return; }

typingIndicator.textContent = ${raw.name} is typing...;

// clear after a while

setTimeout(()=>{ if(typingIndicator.textContent.includes(raw.name)) typingIndicator.textContent=''; }, 3000);

}

// send button

sendBtn.addEventListener('click', async ()=>{

await handleSend();

});

input.addEventListener('keydown', async (e)=>{

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

});

async function handleSend(){

const text = input.value.trim();

let imgData = null;

if(fileInput.files && fileInput.files[0]){

imgData = await fileToDataUrl(fileInput.files[0]);

}

if(!text && !imgData) return;

await sendMessage({text,image:imgData});

input.value='';

}

function fileToDataUrl(file){

return new Promise((res,rej)=>{

const fr = new FileReader(); fr.onload = ()=>res(fr.result); fr.onerror = rej; fr.readAsDataURL(file);

})

}

// simple WebSocket helper

wsConnect.addEventListener('click', ()=>{

const url = wsUrl.value.trim();

if(!url) return alert('Enter a WebSocket URL (ws:// or wss://) if you want networked realtime.');

if(WS && WS.readyState === WebSocket.OPEN){ WS.close(); return; }

WS = new WebSocket(url);

statusEl.textContent = 'connecting...';

WS.addEventListener('open', ()=>{ statusEl.textContent = 'connected (ws)'; wsConnect.textContent='Disconnect'; });

WS.addEventListener('message', (ev)=>{

try{ const data = JSON.parse(ev.data); if(data.type === 'message' && data.payload){ const msg = data.payload; if(msg.channel === channelInput.value) appendMessage(msg,false); }

}catch(e){console.warn('ws msg',e)}

});

WS.addEventListener('close', ()=>{ statusEl.textContent='disconnected'; wsConnect.textContent='Connect'; WS=null; });

WS.addEventListener('error', (e)=>{ statusEl.textContent='error'; console.warn(e);});

});

// initialize

loadHistory();

statusEl.textContent = 'ready';

// accessibility: focus input

input.focus();

</script></body>

</html>