<!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;

--muted: #96a0b8;

--accent: #4f46e5;

--incoming: #111827;

--outgoing: #063f30;

--glass: rgba(255,255,255,0.03);

--radius: 14px;

Font-family: Inter, ui-sans-serif, system-ui, -apple-system, “Segoe UI”, Roboto, “Helvetica Neue”, Arial;

}

\*{box-sizing:border-box}

Html,body{height:100%;margin:0;background:linear-gradient(180deg,#071024 0%, #041021 100%);color:#e6eef8}

.app {

Max-width:1000px;

Margin:28px auto;

Height:82vh;

Display:grid;

Grid-template-columns: 320px 1fr;

Gap:20px;

Padding:18px;

}

.sidebar {

Background: linear-gradient(180deg, rgba(255,255,255,0.02), rgba(255,255,255,0.01));

Border-radius: var(--radius);

Padding:14px;

Display:flex;

Flex-direction:column;

Gap:12px;

Box-shadow: 0 6px 24px rgba(2,6,23,0.6);

}

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

.brand .logo{

Width:44px; height:44px; border-radius:10px; background:linear-gradient(135deg,var(--accent),#06b6d4);

Display:flex; align-items:center; justify-content:center; font-weight:700; color:white; font-size:18px;

}

.search {

Display:flex; gap:8px; align-items:center;

Background:var(--glass); padding:8px;border-radius:10px;

}

.search input{background:transparent;border:0;outline:none;color:var(--muted);width:100%}

.conversations { overflow:auto; padding-right:6px; }

.conv {

Display:flex; gap:10px; align-items:center; padding:10px; border-radius:10px; cursor:pointer;

Transition: background .18s;

}

.conv:hover{background:rgba(255,255,255,0.02)}

.conv.active{background:rgba(79,70,229,0.12)}

.avatar{width:44px;height:44px;border-radius:10px;background:#0b1220;color:#fff;display:flex;align-items:center;justify-content:center;font-weight:600}

.meta {flex:1; display:flex; flex-direction:column;}

.meta .row{display:flex;justify-content:space-between;align-items:center}

.meta .name{font-weight:600}

.meta .snippet{color:var(--muted); font-size:13px; margin-top:6px}

.main {

Background: linear-gradient(180deg, rgba(255,255,255,0.02), rgba(0,0,0,0.03));

Border-radius: var(--radius);

Display:flex; flex-direction:column; overflow:hidden;

Box-shadow: 0 6px 30px rgba(2,6,23,0.6);

}

.header {

Display:flex; align-items:center; gap:12px; padding:12px 16px; border-bottom:1px solid rgba(255,255,255,0.02);

}

.title {flex:1}

.status {color:var(--muted); font-size:13px}

.messages {

Flex:1; padding:18px; overflow:auto; display:flex; flex-direction:column; gap:12px;

Background-image: radial-gradient(rgba(255,255,255,0.002) 1px, transparent 1px);

Background-size: 18px 18px;

}

.msg {

Max-width:70%; padding:10px 12px; border-radius:12px; line-height:1.3; font-size:14px;

Display:inline-block;

Word-wrap:break-word;

}

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

.incoming { background:var(--incoming); border:1px solid rgba(255,255,255,0.02); }

.outgoing { margin-left:auto; background:linear-gradient(180deg,#065f46,#043a2b); color:#d7ffee; border:1px solid rgba(255,255,255,0.03); }

.meta-time { font-size:11px; color:var(--muted); margin-top:4px; }

.composer {

Display:flex; gap:10px; align-items:center; padding:12px; border-top:1px solid rgba(255,255,255,0.02);

Background: linear-gradient(180deg, rgba(255,255,255,0.01), transparent);

}

.input {

Flex:1; display:flex; gap:8px; align-items:center; background:var(--glass); padding:8px; border-radius:12px;

}

.input textarea{flex:1;background:transparent;border:0;outline:none;color:inherit;resize:none;height:44px;padding:6px;font-size:14px}

.btn { background:var(--accent); border:none;color:white;padding:10px 14px;border-radius:10px;cursor:pointer;font-weight:600;box-shadow:0 6px 18px rgba(79,70,229,0.18)}

.tiny { font-size:12px; padding:6px 10px; border-radius:8px; background:transparent; border:1px solid rgba(255,255,255,0.03); color:var(--muted) }

.typing { font-size:13px; color:var(--muted); padding-left:6px }

.status-dot { width:10px;height:10px;border-radius:50%;display:inline-block;margin-right:6px;vertical-align:middle }

.online { background:#10b981; box-shadow:0 0 8px rgba(16,185,129,0.18) }

.offline { background:#6b7280 }

.small { font-size:12px }

.centered {

Display:flex;align-items:center;justify-content:center;height:100%; color:var(--muted); font-size:15px;

Flex-direction:column; gap:10px;

}

@media(max-width:880px){

.app{ grid-template-columns: 1fr; height:90vh; padding:12px;}

.sidebar{order:2;height:180px;overflow:auto}

.main{order:1}

}

</style>

</head>

<body>

<div class=”app” id=”app”>

<div class=”sidebar”>

<div class=”brand”>

<div class=”logo”>RT</div>

<div>

<div style=”font-weight:700”>RealTimeChat</div>

<div class=”small” style=”color:var(--muted)”>Single-file demo</div>

</div>

</div>

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

<div class=”status-dot online” id=”presence-dot”></div>

<div id=”presence-text” class=”small”>Connecting…</div>

<div style=”margin-left:auto”>

<button class=”tiny” id=”modeToggle”>Demo mode</button>

</div>

</div>

<div class=”search”>

<input id=”search” placeholder=”Search conversations (demo)…” />

</div>

<div style=”flex:1; overflow:auto” class=”conversations” id=”convList”>

<!—Dynamic conversation list (demo has one) 🡪

</div>

<div style=”display:flex;gap:8px;align-items:center;justify-content:center;padding-top:6px”>

<div class=”small” style=”color:var(--muted)”>WS [URL:</div](URL:%3c/div)>

<input id=”wsUrl” placeholder=”wss://your-server.example/ws” style=”flex:1;background:transparent;border:1px dashed rgba(255,255,255,0.03);padding:6px;border-radius:8px;color:var(--muted)” />

<button id=”connectBtn” class=”tiny”>Connect</button>

</div>

</div>

<div class=”main” role=”main” aria-live=”polite”>

<div class=”header”>

<div style=”display:flex;gap:12px;align-items:center”>

<div class=”avatar” id=”chatAvatar”>A</div>

</div>

<div class=”title”>

<div style=”font-weight:700” id=”chatName”>Alice (Demo)</div>

<div class=”status” id=”chatStatus”>Tap a conversation to start</div>

</div>

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

<button class=”tiny” id=”clearBtn”>Clear</button>

<button class=”tiny” id=”exportBtn”>Export</button>

</div>

</div>

<div class=”messages” id=”messages” tabindex=”0” aria-label=”Messages”>

<div class=”centered” id=”emptyState”>

No messages yet — send something!<br/><span class=”small” style=”color:var(--muted)”>Demo mode will echo replies automatically.</span>

</div>

</div>

<div class=”composer”>

<div class=”input” style=”min-width:0”>

<textarea id=”input” placeholder=”Type a message…” rows=”1”></textarea>

<div style=”display:flex;flex-direction:column;gap:6px”>

<div class=”typing” id=”typingIndicator” style=”visibility:hidden”>typing…</div>

<div style=”display:flex;gap:6px”>

<button class=”tiny” id=”emojiBtn”>😊</button>

</div>

</div>

</div>

<button class=”btn” id=”sendBtn”>Send</button>

</div>

</div>

</div>

<script>

/\*

Single-file real-time chat demo.

- Set WS\_URL to a websocket server (wss://…) to use live mode.

- Otherwise it runs in demo/offline mode which simulates responses.

\*/

/\* --------- CONFIG --------- \*/

// Default WebSocket URL (leave empty to start in Demo mode)

Let WS\_URL = “”; // e.g. “wss://your-server.example/ws”

// Identity (in a real app this comes from auth)

Const ME = { id: “me-123”, name: “You”, avatar: “Y” };

Const THEM = { id: “them-1”, name: “Alice”, avatar: “A” };

/\* --------- Utilities ---------- \*/

Const $ = (s) => document.querySelector(s);

Const uid = () => ‘m\_’ + Math.random().toString(36).slice(2,10);

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

Const fmtTime = (iso) => {

Const d = new Date(iso);

Return d.toLocaleTimeString([], {hour:’2-digit’,minute:’2-digit’});

};

/\* --------- Persistence (localStorage) ---------- \*/

Const STORAGE\_KEY = “rtchat\_messages\_v1”;

Function loadMessages() {

Try {

Const raw = localStorage.getItem(STORAGE\_KEY);

Return raw ? JSON.parse(raw) : [];

} catch€{ return [];}

}

Function saveMessages(msgs) { localStorage.setItem(STORAGE\_KEY, JSON.stringify(msgs)); }

/\* --------- App State ---------- \*/

Let messages = loadMessages(); // array of message objects

Let ws = null;

Let connected = false;

Let demoMode = !WS\_URL;

Let backoff = 500; // ms

/\* message model:

{

Id: string,

senderId: string,

text: string,

createdAt: ISO,

status: “sending”|”sent”|”delivered”|”read”|”failed”

}

\*/

/\* --------- DOM refs ---------- \*/

Const convList = $(‘#convList’);

Const messagesEl = $(‘#messages’);

Const chatNameEl = $(‘#chatName’);

Const chatStatusEl = $(‘#chatStatus’);

Const presenceDot = $(‘#presence-dot’);

Const presenceText = $(‘#presence-text’);

Const wsUrlInput = $(‘#wsUrl’);

Const connectBtn = $(‘#connectBtn’);

Const modeToggle = $(‘#modeToggle’);

Const inputEl = $(‘#input’);

Const sendBtn = $(‘#sendBtn’);

Const typingIndicator = $(‘#typingIndicator’);

Const clearBtn = $(‘#clearBtn’);

Const exportBtn = $(‘#exportBtn’);

/\* --------- Rendering ---------- \*/

Function renderConversations(){

// demo: single conversation

convList.innerHTML = ‘’;

const div = document.createElement(‘div’);

div.className = ‘conv active’;

div.innerHTML = `

<div class=”avatar”>${THEM.avatar}</div>

<div class=”meta”>

<div class=”row”><div class=”name”>${THEM.name}</div><div class=”small”>${messages.length ? fmtTime(messages[messages.length-1].createdAt) : ‘’}</div></div>

<div class=”snippet”>${messages.length? (messages[messages.length-1].text.slice(0,50)) : ‘No messages yet’}</div>

</div>

`;

convList.appendChild(div);

}

Function renderMessages(){

messagesEl.scrollTop = messagesEl.scrollHeight;

messagesEl.innerHTML = ‘’;

if(messages.length === 0){

$(‘#emptyState’).style.display = ‘flex’;

Return;

} else {

$(‘#emptyState’).style.display = ‘none’;

}

For(const m of messages){

Const row = document.createElement(‘div’);

Row.className = ‘msg-row’;

Const bubble = document.createElement(‘div’);

Bubble.className = ‘msg’;

Bubble.textContent = m.text;

Const meta = document.createElement(‘div’);

Meta.className = ‘meta-time small’;

Meta.innerHTML = `${fmtTime(m.createdAt)} ${m.senderId===ME.id ? ‘• ‘ + (m.status||’’) : ‘’}`;

If(m.senderId === ME.id){

Bubble.classList.add(‘outgoing’);

Row.appendChild(bubble);

Const mcol = document.createElement(‘div’);

Mcol.style.minWidth = ‘90px’;

Mcol.style.textAlign = ‘right’;

Mcol.appendChild(meta);

Row.appendChild(mcol);

} else {

Bubble.classList.add(‘incoming’);

Const avatar = document.createElement(‘div’);

Avatar.className = ‘avatar’;

Avatar.textContent = THEM.avatar;

Row.appendChild(avatar);

Const col = document.createElement(‘div’);

Col.style.display = ‘flex’;

Col.style.flexDirection = ‘column’;

Col.appendChild(bubble);

Col.appendChild(meta);

Row.appendChild(col);

}

messagesEl.appendChild(row);

}

// scroll to bottom

setTimeout(()=> { messagesEl.scrollTop = messagesEl.scrollHeight; }, 10);

}

/\* --------- Message operations ---------- \*/

Function addMessage(text, sender = ME.id, opts = {}) {

Const m = {

Id: uid(),

senderId: sender,

text: text,

createdAt: nowISO(),

status: opts.status || (sender === ME.id ? ‘sending’ : ‘delivered’)

};

Messages.push(m);

saveMessages(messages);

renderMessages();

renderConversations();

return m;

}

Function updateMessageStatus(id, status){

Const m = messages.find(x=>x.id===id);

If(m){

m.status = status;

saveMessages(messages);

renderMessages();

}

}

/\* --------- WebSocket & Networking ---------- \*/

Function setPresence(online){

presenceDot.className = ‘status-dot ‘ + (online ? ‘online’ : ‘offline’);

presenceText.textContent = online ? ‘Online’ : ‘Offline’;

chatStatusEl.textContent = online ? ‘Connected’ : ‘Disconnected’;

}

Function connectWS(url){

If(!url) return;

demoMode = false;

WS\_URL = url;

wsUrlInput.value = url;

presenceText.textContent = ‘Connecting…’;

try {

ws = new WebSocket(url);

} catch(err){

Console.error(‘WS init failed’, err);

goDemo(‘WebSocket init failed’);

return;

}

Ws.addEventListener(‘open’, () => {

Console.log(‘ws open’);

Connected = true;

Backoff = 500;

setPresence(true);

sendPresence();

chatStatusEl.textContent = ‘Connected to ‘ + url;

});

Ws.addEventListener(‘message’, (ev) => {

// Expecting JSON messages with a simple protocol:

// { type: “message”, id, senderId, text, createdAt }

// { type: “ack”, id } -> ack for messages we sent

Try {

Const data = JSON.parse(ev.data);

If(data.type === ‘message’){

// incoming message

addMessage(data.text, data.senderId, { status: ‘delivered’ });

// send read receipt optionally

Ws.send(JSON.stringify({type:’read’, id: data.id}));

} else if (data.type === ‘ack’){

updateMessageStatus(data.id, ‘sent’);

} else if (data.type === ‘status’){

// e.g. presence updates

Console.log(‘status’, data);

} else if (data.type === ‘typing’){

showTyping(true);

// hide later

setTimeout(()=> showTyping(false), 1400);

} else {

Console.log(‘unknown ws payload’, data);

}

} catch€{

Console.log(‘Non-JSON or malformed ws message’, ev.data);

}

});

Ws.addEventListener(‘close’, () => {

Console.log(‘ws close’);

Connected = false;

setPresence(false);

chatStatusEl.textContent = ‘Disconnected’;

// fallback to demo if connection can’t be restored

scheduleReconnect();

});

Ws.addEventListener(‘error’, (err) => {

Console.warn(‘ws error’, err);

Connected = false;

setPresence(false);

scheduleReconnect();

});

}

Let reconnectTimer = null;

Function scheduleReconnect(){

If(demoMode) return;

If(reconnectTimer) return;

Const delay = backoff;

reconnectTimer = setTimeout(() => {

reconnectTimer = null;

backoff = Math.min(60000, backoff \* 1.5);

try {

connectWS(WS\_URL);

} catch€{}

}, delay);

}

/\* --------- Demo mode (local echo) ---------- \*/

Let demoReplyTimer = null;

Function goDemo(reason){

Console.info(‘Switching to demo mode:’, reason);

demoMode = true;

if(ws){

try { ws.close(); } catch€{}

ws = null;

}

Connected = false;

setPresence(false);

chatStatusEl.textContent = ‘Demo mode’;

modeToggle.textContent = ‘Demo mode’;

}

Function sendPresence(){

If(ws && connected){

Ws.send(JSON.stringify({type:’presence’, userId: ME.id, name: ME.name}));

}

}

Function showTyping(on){

typingIndicator.style.visibility = on ? ‘visible’ : ‘hidden’;

}

/\* --------- Sending messages ---------- \*/

Function sendMessage(text){

If(!text || !text.trim()) return;

Const local = addMessage(text, ME.id, { status: ‘sending’ });

// If connected to a real server, send JSON and rely on ack to update status

If(!demoMode && ws && connected){

Try {

Ws.send(JSON.stringify({ type: ‘message’, id: local.id, senderId: ME.id, text: text, createdAt: local.createdAt }));

// simulate ack if server doesn’t respond immediately (timeout)

setTimeout(()=>{ if(local.status === ‘sending’) updateMessageStatus(local.id, ‘sent’); }, 6000);

} catch€{

updateMessageStatus(local.id, ‘failed’);

console.error(‘send failed’, e);

}

} else {

// demo mode: echo back with delay; mark as sent and delivered

updateMessageStatus(local.id, ‘sent’);

// show typing indicator for reply

showTyping(true);

if(demoReplyTimer) clearTimeout(demoReplyTimer);

demoReplyTimer = setTimeout(()=> {

showTyping(false);

const replyText = generateDemoReply(text);

addMessage(replyText, THEM.id, { status: ‘delivered’ });

updateMessageStatus(local.id, ‘delivered’);

}, 800 + Math.random() \* 900);

}

}

/\* --------- Demo reply generator ---------- \*/

Function generateDemoReply(incoming){

Const lc = incoming.toLowerCase();

If(lc.includes(‘hello’) || lc.includes(‘hi’)) return ‘Hi! How can I help?’;

If(lc.includes(‘time’)) return ‘It\’s ‘ + new Date().toLocaleTimeString();

If(lc.includes(‘demo’)) return ‘This is the demo echo bot.’;

If(lc.includes(‘image’)) return ‘Nice image! (demo)’;

// default small echo

Return “Echo: “ + incoming;

}

/\* --------- UI wiring ---------- \*/

sendBtn.addEventListener(‘click’, ()=> {

sendMessage(inputEl.value);

inputEl.value = ‘’;

inputEl.focus();

});

inputEl.addEventListener(‘keydown’, €=>{

if(e.key === ‘Enter’ && !e.shiftKey){

e.preventDefault();

sendBtn.click();

} else {

// notify typing to server

If(!demoMode && ws && connected){

Try { ws.send(JSON.stringify({type:’typing’, userId: ME.id})); } catch€{}

}

}

});

connectBtn.addEventListener(‘click’, ()=> {

const url = wsUrlInput.value.trim();

if(!url){

alert(‘Enter a WebSocket URL to connect (or leave empty for demo).’);

return;

}

goDemo(‘User initiated connect’);

setTimeout(()=> connectWS(url), 100);

});

modeToggle.addEventListener(‘click’, ()=> {

// toggle between demo and live (only toggles UI; does not auto connect)

demoMode = !demoMode;

modeToggle.textContent = demoMode ? ‘Demo mode’ : ‘Live mode’;

chatStatusEl.textContent = demoMode ? ‘Demo mode’ : ‘Enter a WebSocket URL and hit Connect’;

setPresence(!demoMode);

});

clearBtn.addEventListener(‘click’, ()=> {

if(!confirm(‘Clear all local messages?’)) return;

messages = [];

saveMessages(messages);

renderMessages();

renderConversations();

});

exportBtn.addEventListener(‘click’, ()=> {

const data = JSON.stringify(messages, null, 2);

const blob = new Blob([data], {type:’application/json’});

const url = URL.createObjectURL(blob);

const a = document.createElement(‘a’);

a.href = url;

a.download = ‘chat-export.json’;

document.body.appendChild(a);

a.click();

a.remove();

URL.revokeObjectURL(url);

});

/\* ------ Initialize UI ------ \*/

Function init(){

wsUrlInput.value = WS\_URL;

modeToggle.textContent = demoMode ? ‘Demo mode’ : ‘Live mode’;

renderConversations();

renderMessages();

setPresence(false);

chatNam