**USER AUTHENTICATION SYSTEM**

<!doctype html>

<html lang=”en”>

<head>

<meta charset=”utf-8” />

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

<title>Client-side Auth Demo</title>

<style>

:root{

--bg:#0f1724; --card:#0b1220; --muted:#94a3b8; --accent:#60a5fa; --success:#34d399; --danger:#fb7185;

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

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

}

\*{box-sizing:border-box}

Body{

Margin:0; min-height:100vh; display:flex; align-items:center; justify-content:center;

Background:linear-gradient(180deg,#071429 0%, #081026 60%);

Color:#e6eef8;

Padding:24px;

}

.container{

Width:100%; max-width:920px; background:var(--card); border-radius:14px; padding:22px;

Box-shadow: 0 10px 30px rgba(2,6,23,0.6);

Display:grid; grid-template-columns:1fr 420px; gap:18px;

}

.left{

Padding:18px;

}

.brand{font-weight:700; font-size:20px; margin-bottom:8px; color:var(--accent)}

.lead{color:var(--muted); margin-bottom:16px; line-height:1.4}

.features{display:grid; gap:8px; margin-top:6px}

.feature{background:var(--glass); padding:10px; border-radius:10px; color:var(--muted); font-size:14px}

.card{

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

Padding:18px; border-radius:12px;

}

Label{display:block; margin-bottom:6px; font-size:13px; color:var(--muted)}

Input[type=”text”], input[type=”email”], input[type=”password”]{

Width:100%; padding:10px 12px; border-radius:8px; border:1px solid rgba(255,255,255,0.04);

Background:transparent; color:inherit; margin-bottom:12px; font-size:14px;

}

.row{display:flex; gap:10px}

Button{

Border:0; background:var(--accent); color:#07203a; padding:10px 12px; border-radius:10px; cursor:pointer;

Font-weight:600;

}

Button.ghost{background:transparent; color:var(--muted); border:1px solid rgba(255,255,255,0.03)}

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

.error{color:var(--danger); font-size:13px}

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

.switch{display:flex; justify-content:space-between; align-items:center; margin-bottom:12px}

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

.meter{height:8px; background:rgba(255,255,255,0.04); border-radius:8px; overflow:hidden; margin-bottom:10px}

.meter > i{display:block; height:100%; width:0%; transition:width 220ms ease}

.user-list{margin-top:12px; font-size:13px; color:var(--muted)}

.token{font-family:monospace; font-size:12px; background:rgba(255,255,255,0.02); padding:6px 8px; border-radius:8px; display:inline-block}

.center{display:flex; align-items:center; justify-content:center; flex-direction:column; gap:6px}

/\* Responsive \*/

@media (max-width:880px){

.container{grid-template-columns:1fr; padding:16px}

}

</style>

</head>

<body>

<div class=”container” role=”main”>

<div class=”left”>

<div class=”brand”>Auth Demo — Client-side</div>

<div class=”lead”>

A minimal user registration and sign-in demo using the Web Crypto API (PBKDF2 & SHA-256).

Stores user credentials (salt + derivedKey) in <code>localStorage</code>. Session token lives in <code>sessionStorage</code>.

</div>

<div class=”card”>

<div style=”display:flex; justify-content:space-between; align-items:center; margin-bottom:12px”>

<strong>Active session</strong>

<button class=”ghost” id=”btnDemoFill”>Demo user</button>

</div>

<div id=”sessionArea” class=”center” aria-live=”polite”>

<div class=”muted” id=”noSession”>No one is signed in.</div>

<div id=”sessionInfo” style=”display:none; width:100%”>

<div style=”display:flex; justify-content:space-between; align-items:center”>

<div>

<div style=”font-weight:700” id=”who”>—you—</div>

<div class=”small muted” id=”emailShown”></div>

</div>

<div style=”text-align:right”>

<div class=”small muted”>Token</div>

<div class=”token” id=”tokenVal” title=”Session token”></div>

</div>

</div>

<div style=”display:flex; gap:8px; margin-top:12px; justify-content:flex-end”>

<button id=”btnLogout”>Logout</button>

<button class=”ghost” id=”btnClearAll”>Clear all users</button>

</div>

</div>

</div>

<div class=”user-list” id=”userListBox”></div>

</div>

<div class=”features” style=”margin-top:12px”>

<div class=”feature”><strong>Notes:</strong> This demo uses PBKDF2 with a random salt and stores the derived key. Never store raw passwords in real apps.</div>

<div class=”feature”>For production, move authentication to a secure server: store salted/hased passwords (bcrypt/argon2) server-side and use HTTPS + secure cookies / JWTs.</div>

</div>

</div>

<div>

<div class=”card” style=”margin-bottom:12px”>

<div class=”switch”>

<div style=”font-weight:700” id=”formTitle”>Register</div>

<div class=”small muted”>Already registered? <a href=”#” id=”toggleForm”>Sign in</a></div>

</div>

<form id=”authForm” autocomplete=”off”>

<label for=”name”>Name</label>

<input id=”name” type=”text” placeholder=”Your name” required />

<label for=”email”>Email</label>

<input id=”email” type=”email” placeholder=[you@example.com](mailto:you@example.com) required />

<label for=”password”>Password</label>

<input id=”password” type=”password” placeholder=”At least 8 characters” required minlength=”8” />

<div style=”display:flex; gap:8px; align-items:center; margin-bottom:12px”>

<div style=”flex:1”>

<label class=”small muted”>Password strength</label>

<div class=”meter” aria-hidden=”true”><I id=”meterBar”></i></div>

<div class=”small muted” id=”pwNote”>Use 8+ chars, with letters, numbers, and symbols.</div>

</div>

<div>

<label class=”small muted”>&nbsp;</label>

<div style=”display:flex; gap:6px”>

<button type=”button” class=”ghost” id=”btnToggleReveal”>Show</button>

</div>

</div>

</div>

<div style=”display:flex; gap:8px; align-items:center; margin-bottom:12px”>

<label style=”margin:0; display:flex; gap:8px; align-items:center”>

<input id=”remember” type=”checkbox”> <span class=”small muted”>Remember me (keeps token in session)</span>

</label>

</div>

<div style=”display:flex; gap:8px; justify-content:flex-end”>

<button id=”submitBtn” type=”submit”>Register</button>

<button type=”button” class=”ghost” id=”btnReset”>Reset</button>

</div>

<div style=”height:12px; margin-top:10px”>

<div id=”message” role=”status” aria-live=”polite”></div>

</div>

</form>

</div>

<div class=”card” style=”margin-top:0”>

<strong>Developer tools</strong>

<div class=”small muted” style=”margin-top:8px”>Local storage preview (sanitized):</div>

<pre id=”debug” style=”background:rgba(0,0,0,0.2); padding:12px; border-radius:8px; font-size:12px; margin-top:8px; max-height:180px; overflow:auto”></pre>

</div>

</div>

</div>

<script>

/\*

Client-side auth demo

- Uses Web Crypto PBKDF2 (SHA-256) to derive a key from password + random salt

- Stores: { name, email, salt (base64), iterations, derivedKeyHex } in localStorage.users (object)

- Session token stored in sessionStorage.currentToken

NOTE: This is not a secure production design. Demo & learning only.

\*/

// ---------- Helpers ----------

Const $ = id => document.getElementById(id);

Const enc = new TextEncoder();

Const dec = new TextDecoder();

Function toHex(buffer){

Const bytes = new Uint8Array(buffer);

Return Array.from(bytes).map(b => b.toString(16).padStart(2,’0’)).join(‘’);

}

Function toBase64(buffer){

Const bytes = new Uint8Array(buffer);

Let binary = ‘’;

For (let b of bytes) binary += String.fromCharCode(b);

Return btoa(binary);

}

Function fromBase64(b64){

Const bin = atob(b64);

Const len = bin.length;

Const arr = new Uint8Array(len);

For (let i=0;i<len;i++) arr[i]=bin.charCodeAt(i);

Return arr.buffer;

}

Function randomBytes(n){

Const a = new Uint8Array(n);

Crypto.getRandomValues(a);

Return a.buffer;

}

Function safeParseJSON(s, fallback){ try { return JSON.parse(s); } catch€{ return fallback; } }

// ---------- Storage utility ----------

Const STORAGE\_KEY = ‘auth\_demo\_users\_v1’;

Function loadUsers(){

Return safeParseJSON(localStorage.getItem(STORAGE\_KEY), {});

}

Function saveUsers(obj){

localStorage.setItem(STORAGE\_KEY, JSON.stringify(obj));

}

Function clearAllUsers(){

localStorage.removeItem(STORAGE\_KEY);

}

// ---------- Crypto: PBKDF2 derive ----------

Async function deriveKeyPBKDF2(password, saltBuffer, iterations=150000, len=32){

// returns ArrayBuffer of derived bits (len bytes)

Const pwKey = await crypto.subtle.importKey(

‘raw’, enc.encode(password), {name:’PBKDF2’}, false, [‘deriveBits’]

);

Const derived = await crypto.subtle.deriveBits(

{name:’PBKDF2’, hash:’SHA-256’, salt: saltBuffer, iterations},

pwKey,

len \* 8

);

Return derived; // ArrayBuffer

}

// ---------- Auth logic ----------

Async function registerUser(name, email, password){

Const users = loadUsers();

Const key = email.toLowerCase();

If (users[key]) throw new Error(‘Email already registered.’);

// generate salt

Const salt = randomBytes(16);

Const iterations = 150000; // demonstration; choose high but browser-friendly

Const derived = await deriveKeyPBKDF2(password, salt, iterations, 32);

Users[key] = {

Name: name || ‘’,

Email: email,

Salt: toBase64(salt),

Iterations,

Derived: toHex(derived),

createdAt: new Date().toISOString()

};

saveUsers(users);

return true;

}

Async function verifyUser(email, password){

Const users = loadUsers();

Const key = email.toLowerCase();

Const record = users[key];

If (!record) return false;

Const salt = fromBase64(record.salt);

Const derived = await deriveKeyPBKDF2(password, salt, record.iterations, 32);

Const hex = toHex(derived);

Return hex === record.derived;

}

// ---------- Session ----------

Function createToken(){

// simple random token for demo

Const t = toHex(randomBytes(16));

Return t;

}

Function signIn(email){

Const users = loadUsers();

Const u = users[email.toLowerCase()];

If (!u) return;

Const token = createToken();

Const session = { email: u.email, name: u.name, token, signedAt: new Date().toISOString() };

// store token in sessionStorage (cleared when tab closed). If “remember” checked we could persist longer.

sessionStorage.setItem(‘auth\_demo\_session’, JSON.stringify(session));

updateSessionUI();

}

Function signOut(){

sessionStorage.removeItem(‘auth\_demo\_session’);

updateSessionUI();

}

// ---------- UI wiring ----------

Const formTitle = $(‘formTitle’);

Const authForm = $(‘authForm’);

Const toggleForm = $(‘toggleForm’);

Const submitBtn = $(‘submitBtn’);

Const messageBox = $(‘message’);

Const nameInput = $(‘name’);

Const emailInput = $(‘email’);

Const passwordInput = $(‘password’);

Const rememberChk = $(‘remember’);

Const meterBar = $(‘meterBar’);

Const pwNote = $(‘pwNote’);

Const btnToggleReveal = $(‘btnToggleReveal’);

Const debug = $(‘debug’);

Const userListBox = $(‘userListBox’);

Const sessionInfo = $(‘sessionInfo’);

Const sessionArea = $(‘sessionArea’);

Const who = $(‘who’);

Const emailShown = $(‘emailShown’);

Const tokenVal = $(‘tokenVal’);

Const noSession = $(‘noSession’);

Const btnLogout = $(‘btnLogout’);

Const btnClearAll = $(‘btnClearAll’);

Const btnDemoFill = $(‘btnDemoFill’);

Const btnReset = $(‘btnReset’);

Let isRegister = true;

Function renderUsers(){

Const users = loadUsers();

Const keys = Object.keys(users);

If (keys.length === 0){

userListBox.innerHTML = ‘<div class=”small muted”>No registered users</div>’;

debug.textContent = JSON.stringify({}, null, 2);

return;

}

Const items = keys.map(k => {

Const u = users[k];

Return `• ${u.name || ‘-‘} — ${u.email} — created ${new Date(u.createdAt).toLocaleString()}`;

});

userListBox.innerHTML = items.map(I => `<div class=”small muted”>${i}</div>`).join(‘’);

// debug: show sanitized data (no raw derived key? But it’s okay for demo). We’ll show only metadata

Const safe = {};

Keys.forEach(k => {

Const {name,email,createdAt,iterations} = users[k];

Safe[k] = {name,email,createdAt,iterations};

});

Debug.textContent = JSON.stringify(safe, null, 2);

}

Function updateSessionUI(){

Const s = safeParseJSON(sessionStorage.getItem(‘auth\_demo\_session’), null);

If (!s){

sessionInfo.style.display=’none’; noSession.style.display=’block’;

} else {

sessionInfo.style.display=’block’; noSession.style.display=’none’;

who.textContent = s.name || ‘(no name)’;

emailShown.textContent = s.email;

tokenVal.textContent = s.token.slice(0,12)+’…’;

}

renderUsers();

}

toggleForm.addEventListener(‘click’, € => {

e.preventDefault();

isRegister = !isRegister;

formTitle.textContent = isRegister ? ‘Register’ : ‘Sign in’;

submitBtn.textContent = isRegister ? ‘Register’ : ‘Sign in’;

toggleForm.textContent = isRegister ? ‘Sign in’ : ‘Register’;

messageBox.textContent = ‘’;

});

btnToggleReveal.addEventListener(‘click’, € => {

if (passwordInput.type === ‘password’){ passwordInput.type=’text’; btnToggleReveal.textContent=’Hide’; }

else { passwordInput.type=’password’; btnToggleReveal.textContent=’Show’; }

});

passwordInput.addEventListener(‘input’, () => {

const val = passwordInput.value;

let score = 0;

if (val.length >= 8) score += 1;

if (/[A-Z]/.test(val) && /[a-z]/.test(val)) score += 1;

if (/\d/.test(val)) score += 1;

if (/[^A-Za-z0-9]/.test(val)) score += 1;

const pct = Math.min(100, (score / 4) \* 100);

meterBar.style.width = pct + ‘%’;

// color

Const color = score <=1 ? ‘rgba(251,113,133,0.95)’ : score === 2 ? ‘orange’ : score === 3 ? ‘lightgreen’ : ‘var(--success)’;

meterBar.style.background = color;

pwNote.textContent = score >= 3 ? ‘Strong password’ : ‘Use a mix of upper/lowercase, numbers, symbols.’;

});

authForm.addEventListener(‘submit’, async € => {

e.preventDefault();

messageBox.textContent = ‘’;

const name = nameInput.value.trim();

const email = emailInput.value.trim().toLowerCase();

const password = passwordInput.value;

if (!email || !password) { messageBox.innerHTML = ‘<span class=”error”>Please provide email and password.</span>’; return; }

submitBtn.disabled = true;

submitBtn.textContent = isRegister ? ‘Registering…’ : ‘Signing in…’;

try {

if (isRegister){

// register flow

If (password.length < 8) throw new Error(‘Password too short (min 8 chars).’);

Await registerUser(name, email, password);

messageBox.innerHTML = ‘<span class=”success”>Registered! You are signed in now.</span>’;

signIn(email);

// clear password

passwordInput.value = ‘’;

meterBar.style.width = ‘0%’;

} else {

// sign in flow

Const ok = await verifyUser(email, password);

If (!ok) { messageBox.innerHTML = ‘<span class=”error”>Invalid credentials.</span>’; return; }

messageBox.innerHTML = ‘<span class=”success”>Signed in successfully.</span>’;

signIn(email);

passwordInput.value = ‘’;

}

} catch (err){

messageBox.innerHTML = ‘<span class=”error”>’ + (err.message || ‘Error’) + ‘</span>’;

} finally {

submitBtn.disabled = false;

submitBtn.textContent = isRegister ? ‘Register’ : ‘Sign in’;

updateSessionUI();

}

});

// Demo fill button

btnDemoFill.addEventListener(‘click’, async () => {

// create a demo user if not present

Const demoEmail = ‘demo@example.com’;

Const users = loadUsers();

If (!users[demoEmail]){

Try{

Await registerUser(‘Demo User’, demoEmail, ‘Demo1234!’);

messageBox.innerHTML = ‘<span class=”success”>Demo user created: [demo@example.com](mailto:demo@example.com) / Demo1234!</span>’;

}catch€{

Console.warn€;

}

} else {

messageBox.innerHTML = ‘<span class=”muted”>Demo user already exists.</span>’;

}

updateSessionUI();

});

// Logout button

btnLogout.addEventListener(‘click’, € => {

signOut();

messageBox.innerHTML = ‘<span class=”muted”>Logged out.</span>’;

});

// Clear all users (dangerous)

btnClearAll.addEventListener(‘click’, € => {

if (!confirm(‘Delete ALL registered users from localStorage? This cannot be undone.’)) return;

clearAllUsers();

signOut();

messageBox.innerHTML = ‘<span class=”muted”>All users removed.</span>’;

updateSessionUI();

});

// Reset form

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

authForm.reset();

meterBar.style.width = ‘0%’;

messageBox.textContent = ‘’;

});

// on load

updateSessionUI();

</script>

</body>

</html>