**USER AUTHENTICATION SYSTEM**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8" />

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

<title>User Authentication Demo — Single File</title>

<style>

:root{

--bg:#0f1724; --card:#0b1220; --accent:#06b6d4; --muted:#94a3b8; --glass: rgba(255,255,255,0.03);

}

\*{box-sizing:border-box}

body{

margin:0; font-family:Inter,ui-sans-serif,system-ui,-apple-system,"Segoe UI",Roboto,"Helvetica Neue",Arial;

background: linear-gradient(180deg,#071229 0%, #071b2a 100%); color:#e6eef6;

min-height:100vh; display:flex; align-items:center; justify-content:center; padding:32px;

}

.container{

width:100%; max-width:980px; background:linear-gradient(180deg, rgba(255,255,255,0.02), rgba(255,255,255,0.01));

border-radius:14px; padding:20px; box-shadow: 0 8px 30px rgba(2,6,23,0.7);

display:grid; grid-template-columns: 1fr 380px; gap:20px; align-items:start;

}

.left{

padding:18px; background:var(--card); border-radius:10px;

}

h1{margin:0 0 8px 0; font-size:20px}

p.lead{color:var(--muted); margin:0 0 18px 0}

.form{

background:var(--glass); padding:14px; border-radius:8px; margin-bottom:14px;

border:1px solid rgba(255,255,255,0.02)

}

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

input[type="text"], input[type="password"], input[type="email"]{

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

background:transparent; color:inherit; outline:none; margin-bottom:10px;

}

.btn{display:inline-block; padding:10px 12px; border-radius:8px; background:var(--accent); color:#042024; font-weight:600; cursor:pointer; border:none}

.btn.ghost{background:transparent; color:var(--accent); border:1px solid rgba(6,182,212,0.12)}

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

small{color:var(--muted)}

.right{

padding:18px; background:linear-gradient(180deg, rgba(255,255,255,0.01), rgba(255,255,255,0.00));

border-radius:10px; border:1px solid rgba(255,255,255,0.02);

min-height:240px;

}

.card{

background:rgba(255,255,255,0.02); padding:12px; border-radius:8px; margin-bottom:12px;

}

.hidden{display:none}

.row{display:flex; gap:8px}

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

.success{color:#7ee787}

.danger{color:#ff8b8b}

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

nav a{color:var(--muted); text-decoration:none; padding:6px 8px; display:inline-block}

nav a.active{color:var(--accent); font-weight:600}

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

.pw-strength{height:8px; border-radius:6px; background:rgba(255,255,255,0.04); overflow:hidden}

.pw-bar{height:100%; width:0%; transition:width .2s ease}

.log{font-family:monospace; font-size:12px; color:var(--muted); background:rgba(255,255,255,0.01); padding:8px; border-radius:6px; max-height:150px; overflow:auto}

</style>

</head>

<body>

<div class="container" id="app">

<div class="left">

<h1>Auth Demo (client-only)</h1>

<p class="lead">A small single-file demo: register, login, protected dashboard, change password. Passwords are hashed with PBKDF2 + random salt using Web Crypto. Data persists in localStorage (demo only).</p>

<nav id="tabs" style="margin-bottom:12px">

<a href="#" id="tab-login" class="active">Login</a>

<a href="#" id="tab-register">Register</a>

<a href="#" id="tab-dashboard">Dashboard</a>

</nav>

<!-- LOGIN -->

<div id="login" class="form">

<label for="login-identity">Username or Email</label>

<input id="login-identity" type="text" placeholder="username or email" autocomplete="username">

<label for="login-password">Password</label>

<input id="login-password" type="password" placeholder="password" autocomplete="current-password">

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

<button id="btn-login" class="btn">Log in</button>

<button id="btn-demo-user" class="btn ghost">Fill demo user</button>

</div>

<p class="muted" style="margin-top:10px">No account? <a href="#" id="link-to-register">create one</a>.</p>

</div>

<!-- REGISTER -->

<div id="register" class="form hidden">

<label for="reg-username">Username</label>

<input id="reg-username" type="text" placeholder="username (unique)" autocomplete="username">

<label for="reg-email">Email</label>

<input id="reg-email" type="email" placeholder="you@example.com" autocomplete="email">

<label for="reg-password">Password</label>

<input id="reg-password" type="password" placeholder="choose a strong password" autocomplete="new-password">

<label for="reg-password2">Confirm password</label>

<input id="reg-password2" type="password" placeholder="confirm password">

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

<div style="flex:1">

<small class="small-muted">Strength</small>

<div class="pw-strength"><div id="pw-bar" class="pw-bar"></div></div>

</div>

</div>

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

<button id="btn-register" class="btn">Create account</button>

<button id="btn-clear-users" class="btn ghost">Clear demo data</button>

</div>

<p class="muted" style="margin-top:10px">By using this demo you accept that data is stored in localStorage of your browser.</p>

</div>

<!-- MESSAGES / LOG -->

<div class="card">

<div id="message" class="info">No messages yet.</div>

<div style="margin-top:10px"><small class="small-muted">Debug log (localStorage contents):</small></div>

<pre id="log" class="log"></pre>

</div>

<footer>Demo — not production. Password hashing performed locally using PBKDF2. Do not store sensitive credentials in browser-only systems.</footer>

</div>

<div class="right">

<div id="dashboard" class="card hidden">

<h3 id="welcome">Welcome</h3>

<p class="muted" id="user-meta">—</p>

<div style="margin-top:12px;">

<button id="btn-logout" class="btn ghost">Log out</button>

</div>

<hr style="margin:12px 0; border:none; border-top:1px solid rgba(255,255,255,0.02)">

<div>

<h4 style="margin:4px 0 8px 0">Change password</h4>

<label>Current password</label>

<input id="cur-pw" type="password" placeholder="">

<label>New password</label>

<input id="new-pw" type="password" placeholder="">

<label>Confirm new password</label>

<input id="new-pw2" type="password" placeholder="">

<div style="margin-top:8px">

<button id="btn-change-pw" class="btn">Change password</button>

</div>

</div>

</div>

<div class="card">

<h4 style="margin:6px 0">How it works (short)</h4>

<ol style="margin:8px 0 0 18px; color:var(--muted)">

<li>Register: password hashed with PBKDF2 + random salt, stored in localStorage.</li>

<li>Login: entered password re-derived with stored salt and compared.</li>

<li>Session: random token stored in sessionStorage to allow access to dashboard.</li>

</ol>

</div>

<div class="card">

<h4 style="margin:6px 0">Demo user</h4>

<p class="muted" style="margin:6px 0">Click <strong>Fill demo user</strong> (Login) to try an example user, or register your own.</p>

</div>

</div>

</div>

<script>

/\*

Minimal client-side auth demo.

- Uses Web Crypto PBKDF2 with SHA-256 to derive a key from password + salt.

- Stores user objects in localStorage under key "auth\_users" as JSON.

- Session stored in sessionStorage "auth\_session" with { username, token, createdAt }.

NOTE: This is strictly a demo and NOT safe for production. There is no server-side storage/verification.

\*/

const apps = {

usersKey: 'auth\_users',

sessionKey: 'auth\_session',

pbkdf2Iterations: 150000, // decent for demo (higher = slower)

derivedKeyLen: 256 // bits

};

// ----------------- UTIL: encoding -----------------

const enc = new TextEncoder();

const dec = new TextDecoder();

function toBase64(buffer){

// buffer: ArrayBuffer

const bytes = new Uint8Array(buffer);

let binary = '';

for (let i=0;i<bytes.byteLength;i++){

binary += String.fromCharCode(bytes[i]);

}

return btoa(binary);

}

function fromBase64(b64){

const binary = atob(b64);

const len = binary.length;

const bytes = new Uint8Array(len);

for (let i=0;i<len;i++) bytes[i]=binary.charCodeAt(i);

return bytes.buffer;

}

function randBytes(len){

const b = new Uint8Array(len);

crypto.getRandomValues(b);

return b.buffer;

}

function nowIso(){ return new Date().toISOString(); }

// ----------------- CRYPTO: PBKDF2 derive & compare -----------------

async function deriveKeyBase64(password, saltBuffer){

// returns base64 string of derived key

const keyMaterial = await crypto.subtle.importKey(

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

);

const derivedBits = await crypto.subtle.deriveBits(

{ name: 'PBKDF2', salt: saltBuffer, iterations: apps.pbkdf2Iterations, hash:'SHA-256' },

keyMaterial, apps.derivedKeyLen

);

return toBase64(derivedBits);

}

// ----------------- STORAGE: users/session -----------------

function loadUsers(){

const raw = localStorage.getItem(apps.usersKey);

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

}

function saveUsers(list){

localStorage.setItem(apps.usersKey, JSON.stringify(list, null, 2));

}

function saveSession(session){

sessionStorage.setItem(apps.sessionKey, JSON.stringify(session));

}

function loadSession(){

const raw = sessionStorage.getItem(apps.sessionKey);

return raw ? JSON.parse(raw) : null;

}

function clearSession(){

sessionStorage.removeItem(apps.sessionKey);

}

// ----------------- UI helpers -----------------

const el = id => document.getElementById(id);

function show(msg, type='info'){

const box = el('message');

box.className = type==='info' ? 'info' : (type==='ok' ? 'success' : 'danger');

box.textContent = msg;

}

function updateLog(){

const users = loadUsers();

const s = {

usersCount: users.length,

users: users.map(u=>({username:u.username, email:u.email, salt:u.salt, created:u.createdAt}))

};

el('log').textContent = JSON.stringify(s, null, 2);

}

// ----------------- NAV / TABS -----------------

function setActiveTab(tab){

['tab-login','tab-register','tab-dashboard'].forEach(id=>{

el(id).classList.remove('active');

});

el('login').classList.add('hidden');

el('register').classList.add('hidden');

el('dashboard').classList.add('hidden');

if(tab==='login'){

el('tab-login').classList.add('active');

el('login').classList.remove('hidden');

} else if(tab==='register'){

el('tab-register').classList.add('active');

el('register').classList.remove('hidden');

} else if(tab==='dashboard'){

el('tab-dashboard').classList.add('active');

// show dashboard only if authenticated

const session = loadSession();

if(session && session.username){

el('dashboard').classList.remove('hidden');

populateDashboard(session.username);

} else {

show('You must be logged in to view the dashboard.', 'danger');

// fallback to login

setActiveTab('login');

return;

}

}

}

// ----------------- REGISTRATION -----------------

async function registerUser(username, email, password){

username = (username||'').trim();

email = (email||'').trim().toLowerCase();

if(!username || !email || !password){

show('All fields are required for registration.', 'danger'); return;

}

const users = loadUsers();

if(users.some(u=>u.username.toLowerCase() === username.toLowerCase())){

show('Username taken. Choose another.', 'danger'); return;

}

if(users.some(u=>u.email === email)){

show('Email already registered.', 'danger'); return;

}

const salt = randBytes(16);

const saltB64 = toBase64(salt);

const hashB64 = await deriveKeyBase64(password, salt);

const user = { username, email, salt: saltB64, hash: hashB64, createdAt: nowIso() };

users.push(user);

saveUsers(users);

updateLog();

show('Account created. You can now log in.', 'ok');

// auto-fill login form

el('login-identity').value = username;

el('login-password').value = '';

setActiveTab('login');

}

// ----------------- LOGIN -----------------

async function login(identity, password){

identity = (identity||'').trim();

if(!identity || !password){ show('Enter username/email and password.', 'danger'); return; }

const users = loadUsers();

const user = users.find(u => u.username.toLowerCase() === identity.toLowerCase() || u.email === identity.toLowerCase());

if(!user){ show('User not found.', 'danger'); return; }

const saltBuf = fromBase64(user.salt);

const derived = await deriveKeyBase64(password, saltBuf);

if(derived === user.hash){

// create session token

const token = toBase64(randBytes(24));

const session = { username: user.username, token, createdAt: nowIso() };

saveSession(session);

show('Login successful.', 'ok');

setActiveTab('dashboard');

} else {

show('Incorrect password.', 'danger');

}

}

// ----------------- DASHBOARD / CHANGE PWD -----------------

function populateDashboard(username){

const users = loadUsers();

const user = users.find(u => u.username===username);

if(!user){ show('Authenticated user not found.', 'danger'); clearSession(); setActiveTab('login'); return; }

el('welcome').textContent = `Hello, ${user.username}!`;

el('user-meta').textContent = `Email: ${user.email} · Account created: ${user.createdAt}`;

}

async function changePassword(username, currentPw, newPw, newPw2){

if(!currentPw || !newPw || !newPw2){ show('All fields required.', 'danger'); return; }

if(newPw !== newPw2){ show('New passwords do not match.', 'danger'); return; }

const users = loadUsers();

const i = users.findIndex(u=>u.username===username);

if(i === -1){ show('User not found.', 'danger'); return; }

const u = users[i];

const saltBuf = fromBase64(u.salt);

const curDerived = await deriveKeyBase64(currentPw, saltBuf);

if(curDerived !== u.hash){ show('Current password incorrect.', 'danger'); return; }

// create new salt + hash

const newSalt = randBytes(16);

const newSaltB64 = toBase64(newSalt);

const newHash = await deriveKeyBase64(newPw, newSalt);

users[i].salt = newSaltB64;

users[i].hash = newHash;

saveUsers(users);

updateLog();

show('Password changed successfully. You will need to log in again.', 'ok');

clearSession();

setActiveTab('login');

}

// ----------------- DEMO / UTILITY -----------------

function clearAllData(){

if(!confirm('Clear all demo user data from localStorage? This cannot be undone.')) return;

localStorage.removeItem(apps.usersKey);

updateLog();

show('Demo data cleared.', 'ok');

}

function fillDemoUser(){

// Create demo if not exists, then fill login fields

const users = loadUsers();

const demoUsername = 'alice';

const demoEmail = 'alice@example.com';

if(!users.some(u => u.username === demoUsername)){

// create demo with a known password "DemoPass123!"

(async ()=>{

await registerUser(demoUsername, demoEmail, 'DemoPass123!');

el('login-identity').value = demoUsername;

el('login-password').value = 'DemoPass123!';

})();

} else {

el('login-identity').value = demoUsername;

el('login-password').value = 'DemoPass123!';

}

}

// ----------------- PASSWORD STRENGTH (simple) -----------------

function assessStrength(pw){

let score = 0;

if(pw.length>=8) score++;

if(/[A-Z]/.test(pw)) score++;

if(/[a-z]/.test(pw)) score++;

if(/\d/.test(pw)) score++;

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

return score; // 0-5

}

function updateStrengthBar(){

const pw = el('reg-password').value || '';

const s = assessStrength(pw);

const pct = (s/5)\*100;

const bar = el('pw-bar');

bar.style.width = pct+'%';

// color based on width

if(pct<40) bar.style.background = '#ff6b6b';

else if(pct<70) bar.style.background = '#f6c85f';

else bar.style.background = '#7ee787';

}

// ----------------- EVENT BINDINGS -----------------

document.addEventListener('DOMContentLoaded', ()=>{

// tabs

el('tab-login').addEventListener('click', e=>{ e.preventDefault(); setActiveTab('login'); });

el('tab-register').addEventListener('click', e=>{ e.preventDefault(); setActiveTab('register'); });

el('tab-dashboard').addEventListener('click', e=>{ e.preventDefault(); setActiveTab('dashboard'); });

el('link-to-register').addEventListener('click', e=>{ e.preventDefault(); setActiveTab('register'); });

el('btn-demo-user').addEventListener('click', e=>{ fillDemoUser(); show('Demo user filled. Click Log in.', 'info'); });

// register action

el('btn-register').addEventListener('click', async ()=>{

const u = el('reg-username').value;

const e = el('reg-email').value;

const p = el('reg-password').value;

const p2 = el('reg-password2').value;

if(p !== p2){ show('Passwords do not match.', 'danger'); return; }

await registerUser(u,e,p);

});

el('btn-clear-users').addEventListener('click', e=>{ clearAllData(); });

// login action

el('btn-login').addEventListener('click', async ()=>{

const id = el('login-identity').value;

const pw = el('login-password').value;

await login(id,pw);

updateLog();

});

// logout

el('btn-logout').addEventListener('click', ()=>{

clearSession();

show('Logged out.', 'info');

setActiveTab('login');

});

// change password

el('btn-change-pw').addEventListener('click', async ()=>{

const session = loadSession();

if(!session){ show('Not authenticated.', 'danger'); setActiveTab('login'); return; }

const cur = el('cur-pw').value;

const nw = el('new-pw').value;

const nw2 = el('new-pw2').value;

await changePassword(session.username, cur, nw, nw2);

});

// pass strength bar

el('reg-password').addEventListener('input', updateStrengthBar);

// quick enter: allow Enter to submit on login/register fields

['login-identity','login-password'].forEach(id=>{

el(id).addEventListener('keypress', e => { if(e.key==='Enter') el('btn-login').click(); });

});

['reg-username','reg-email','reg-password','reg-password2'].forEach(id=>{

el(id).addEventListener('keypress', e => { if(e.key==='Enter') el('btn-register').click(); });

});

// if already logged in, go to dashboard

const session = loadSession();

updateLog();

if(session && session.username){

setActiveTab('dashboard');

} else {

setActiveTab('login');

}

});

</script>

</body>

</html>