<!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8” />

<meta name=”viewport” content=”width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no” />

<title>Login / Register / Post</title>

<style>

/\* Reset and base \*/

* {

Box-sizing: border-box;

}

Body, html {

Margin: 0; padding: 0; height: 100%;

Font-family: ‘Segoe UI’, Tahoma, Geneva, Verdana, sans-serif;

Background: linear-gradient(135deg, #667eea, #764ba2);

Color: #333;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

Display: flex;

Justify-content: center;

Align-items: center;

Min-height: 600px;

Min-width: 350px;

Padding: 20px;

}

.container {

Max-width: 350px;

Width: 100%;

Max-height: 600px;

Background: rgba(255,255,255,0.15);

Border-radius: 15px;

Box-shadow: 0 10px 30px rgba(0,0,0,0.2);

Backdrop-filter: blur(12px);

Color: #fff;

Padding: 25px;

Display: flex;

Flex-direction: column;

Box-sizing: border-box;

}

H2 {

Color: #fff;

Text-shadow: 0 0 5px rgba(0,0,0,0.4);

Font-weight: 700;

Margin-bottom: 20px;

Text-align: center;

}

.alert {

Padding: 12px;

Margin-bottom: 14px;

Border-radius: 8px;

Font-weight: 600;

Font-size: 0.9rem;

Text-align: center;

}

.alert-success {

Background: #7dd57d;

Color: #1f3c17;

Box-shadow: 0 0 12px #6bc87b88;

}

.alert-error {

Background: #e56a6a;

Color: #5a0f0f;

Box-shadow: 0 0 12px #e56a6a88;

}

/\* Responsive for small screens \*/

@media (max-width: 400px) {

Body, html {

Padding: 10px;

Min-width: auto;

}

.container {

Max-width: 100%;

Max-height: none;

Min-height: 600px;

Border-radius: 10px;

Padding: 20px;

}

}

</style>

</head>

<body>

<div class=”container” id=”app-container”>

<!—Dynamic content will be rendered here 🡪

</div>

<script type=”module” src=”./main.js”></script>

</body>

</html>

</content>

</create\_file>

<create\_file>

<path>auth.js</path>

<content>

// auth.js

// Handles login and registration UI, validation, storage, alert messages

Const LS\_KEY\_USERS = ‘users-db’;

Const LS\_KEY\_LOGGED\_IN = ‘logged-in-user’;

/\*\*

\* SHA-256 hashing utility, returns hex string

\* @param {string} message

\* @returns {Promise<string>}

\*/

Async function sha256(message) {

Const msgBuffer = new TextEncoder().encode(message);

Const hashBuffer = await crypto.subtle.digest(‘SHA-256’, msgBuffer);

Const hashArray = Array.from(new Uint8Array(hashBuffer));

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

}

/\*\*

\* Load users object from localStorage

\* @returns {Object} users keyed by email

\*/

Function loadUsers() {

Try {

Return JSON.parse(localStorage.getItem(LS\_KEY\_USERS)) || {};

} catch {

Return {};

}

}

/\*\*

\* Save users object to localStorage

\* @param {Object} users

\*/

Function saveUsers(users) {

localStorage.setItem(LS\_KEY\_USERS, JSON.stringify(users));

}

/\*\*

\* Get currently logged-in user email

\* @returns {string|null}

\*/

Function getLoggedInUser() {

Return localStorage.getItem(LS\_KEY\_LOGGED\_IN);

}

/\*\*

\* Set logged-in user email

\* @param {string} email

\*/

Function setLoggedInUser(email) {

localStorage.setItem(LS\_KEY\_LOGGED\_IN, email);

}

/\*\*

\* Remove logged-in user (logout)

\*/

Function clearLoggedInUser() {

localStorage.removeItem(LS\_KEY\_LOGGED\_IN);

}

/\*\*

\* Creates the login/register form HTML string

\* @param {boolean} isLoginMode

\* @returns {string}

\*/

Function renderForm(isLoginMode = true) {

Return `

<h2>${isLoginMode ? ‘Login’ : ‘Register’}</h2>

<div id=”alert-container”></div>

<form id=”auth-form” autocomplete=”off” novalidate>

<div class=”form-group” style=”${isLoginMode ? ‘display:none;’ : ‘display:block;’ }”>

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

<input type=”text” id=”reg-username” name=”reg-username” autocomplete=”username” placeholder=”Enter username” ${isLoginMode ? ‘’ : ‘required’} />

</div>

<div class=”form-group”>

<label for=”auth-email”>Email</label>

<input type=”email” id=”auth-email” name=”auth-email” autocomplete=”email” placeholder=”Enter email” required />

</div>

<div class=”form-group”>

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

<input type=”password” id=”auth-password” name=”auth-password” autocomplete=”${isLoginMode ? ‘current-password’ : ‘new-password’}” placeholder=”Enter password” required />

</div>

<button type=”submit”>${isLoginMode ? ‘Login’ : ‘Register’}</button>

</form>

<div class=”toggle-link” id=”toggle-auth”>${isLoginMode ? “Don’t have an account? Register” : “Already have an account? Login”}</div>

`;

}

/\*\*

\* Shows an alert message inside container

\* @param {HTMLElement} container

\* @param {string} message

\* @param {‘success’|’error’} type

\*/

Function showAlert(container, message, type=’success’) {

Container.innerHTML = ‘’;

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

div.className = ‘alert ‘ + (type===’error’?’alert-error’:’alert-success’);

div.textContent = message;

container.appendChild(div);

// Remove after 3.5 seconds

setTimeout(() => {

if(container.contains(div)) container.removeChild(div);

}, 3500);

}

/\*\*

\* Initializes the auth UI and logic inside given container

\* @param {HTMLElement} container

\* @param {function} onAuthSuccess Callback(email, username)

\*/

Function initAuth(container, onAuthSuccess) {

// Initial mode login

Let isLoginMode = true;

Function render() {

Container.innerHTML = renderForm(isLoginMode);

// Elements

Const authForm = container.querySelector(‘#auth-form’);

Const toggleLink = container.querySelector(‘#toggle-auth’);

Const alertContainer = container.querySelector(‘#alert-container’);

Const regUsernameInput = container.querySelector(‘#reg-username’);

Const emailInput = container.querySelector(‘#auth-email’);

Const passwordInput = container.querySelector(‘#auth-password’);

Const submitBtn = authForm.querySelector(‘button[type=”submit”]’);

toggleLink.onclick = () => {

isLoginMode = !isLoginMode;

render();

};

authForm.onsubmit = async € => {

e.preventDefault();

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

const password = passwordInput.value.trim();

if (isLoginMode) {

// Login

If (!email || !password) {

showAlert(alertContainer, ‘Please fill all required fields.’, ‘error’);

return;

}

Const users = loadUsers();

If (!users[email]) {

showAlert(alertContainer, ‘No account with that email. Please register.’, ‘error’);

return;

}

Const user = users[email];

Const passHash = await sha256(password);

If (passHash !== user.passwordHash) {

showAlert(alertContainer, ‘Incorrect password. Try again.’, ‘error’);

return;

}

setLoggedInUser(email);

showAlert(alertContainer, ‘Login successful! Redirecting…’, ‘success’);

setTimeout(() => onAuthSuccess(email, user.username), 1200);

} else {

// Register

Const username = regUsernameInput.value.trim();

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

showAlert(alertContainer, ‘Please fill all required fields.’, ‘error’);

return;

}

If (username.length < 3) {

showAlert(alertContainer, ‘Username must be at least 3 characters.’, ‘error’);

return;

}

Const users = loadUsers();

If (users[email]) {

showAlert(alertContainer, ‘Email already registered. Please login.’, ‘error’);

return;

}

Const passwordHash = await sha256(password);

Users[email] = { username, email, passwordHash };

saveUsers(users);

setLoggedInUser(email);

showAlert(alertContainer, ‘Registration successful! Redirecting…’, ‘success’);

setTimeout(() => onAuthSuccess(email, username), 1200);

}

};

}

Render();

Return {

toggleMode() {

isLoginMode = !isLoginMode;

render();

},

clearSession() {

clearLoggedInUser();

},

getLoggedInUser() {

return getLoggedInUser();

}

};

}

Export { initAuth, getLoggedInUser, clearLoggedInUser };

</content>

</create\_file>

<create\_file>

<path>post.js</path>

<content>

// post.js

// Handles posting UI, hashing post content, and output

/\*\*

\* SHA-256 hashing utility, returns hex string

\* @param {string} message

\* @returns {Promise<string>}

\*/

Async function sha256(message) {

Const msgBuffer = new TextEncoder().encode(message);

Const hashBuffer = await crypto.subtle.digest(‘SHA-256’, msgBuffer);

Const hashArray = Array.from(new Uint8Array(hashBuffer));

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

}

/\*\*

\* Creates HTML for the posting form

\* @param {string} username Optional username to greet user

\* @returns {string}

\*/

Function renderPostingPage(username=’’) {

Const greet = username ? `<p style=”margin-bottom:15px; font-weight:600; color:#cfcce0; font-size:1rem; text-align:center;”>Welcome <strong>${escapeHtml(username)}</strong>! Create a new post below.</p>` : ‘’;

Return `

<h2>New Post</h2>

${greet}

<div id=”alert-container”></div>

<form id=”post-form” autocomplete=”off” novalidate>

<div class=”form-group”>

<label for=”post-title”>Post Title</label>

<input type=”text” id=”post-title” name=”post-title” placeholder=”Enter title” required />

</div>

<div class=”form-group”>

<label for=”post-content”>Post Content</label>

<textarea id=”post-content” name=”post-content” placeholder=”Write your content here…” required></textarea>

</div>

<button type=”submit”>Submit Post</button>

</form>

<pre id=”post-hash-output” style=”display:none; background:#2f2d38; color:#a6a6e9; padding:12px; border-radius:10px; font-family: monospace; font-size: 0.9rem; user-select: all; white-space: pre-wrap;”></pre>

<button id=”logout-btn” style=”margin-top:20px; background:#bb4949; border:none; border-radius:10px; padding:10px; font-weight:bold; width: 100%; cursor:pointer;”>Logout</button>

`;

}

/\*\*

\* Escape HTML entities to prevent injection

\* @param {string} unsafe

\* @returns {string}

\*/

Function escapeHtml(unsafe) {

Return unsafe.replace(/[&<”’>]/g, m => ({

‘&’: ‘&amp;’,

‘<’: ‘&lt;’,

‘>’: ‘&gt;’,

‘”’: ‘&quot;’,

“’”: ‘&#39;’

})[m]);

}

/\*\*

\* Shows alert message inside given container

\* @param {HTMLElement} container

\* @param {string} message

\* @param {‘success’|’error’} type

\*/

Function showAlert(container, message, type=’success’) {

Container.innerHTML = ‘’;

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

div.className = ‘alert ‘ + (type===’error’?’alert-error’:’alert-success’);

div.textContent = message;

container.appendChild(div);

// Auto remove after 3.5 sec

setTimeout(() => {

if(container.contains(div)) container.removeChild(div);

}, 3500);

}

/\*\*

\* Initialize the posting page inside given container

\* @param {HTMLElement} container

\* @param {string} username

\* @param {function} onLogout Callback on logout

\*/

Function initPost(container, username, onLogout) {

Container.innerHTML = renderPostingPage(username);

Const postForm = container.querySelector(‘#post-form’);

Const hashOutput = container.querySelector(‘#post-hash-output’);

Const alertContainer = container.querySelector(‘#alert-container’);

Const logoutBtn = container.querySelector(‘#logout-btn’);

postForm.onsubmit = async € => {

e.preventDefault();

const title = postForm.elements[‘post-title’].value.trim();

const content = postForm.elements[‘post-content’].value.trim();

if (!title || !content) {

showAlert(alertContainer, ‘Please enter both post title and content.’, ‘error’);

return;

}

Const toHash = `Title: ${title}\nContent: ${content}`;

Const hash = await sha256(toHash);

hashOutput.textContent = `SHA-256 Hash of your post:\n${hash}`;

hashOutput.style.display = ‘block’;

showAlert(alertContainer, ‘Post submitted and hashed!’, ‘success’);

postForm.reset();

};

logoutBtn.onclick = () => {

if (typeof onLogout === ‘function’) {

onLogout();

}

};

}

Export { initPost };

</content>

</create\_file>

<create\_file>

<path>main.js</path>

<content>

// main.js – App entry point and controller

Import { initAuth, clearLoggedInUser, getLoggedInUser } from ‘./auth.js’;

Import { initPost } from ‘./post.js’;

Const container = document.getElementById(‘app-container’);

/\*\*

\* Shows the login/register view

\*/

Function showAuthView() {

// Initialize the auth module inside container

initAuth(container, (email, username) => {

showPostView(email, username);

});

}

/\*\*

\* Shows posting form view with user greeted

\* @param {string} email

\* @param {string} username

\*/

Function showPostView(email, username) {

initPost(container, username || email, () => {

logout();

});

}

/\*\*

\* Logout the user and show auth view

\*/

Function logout() {

clearLoggedInUser();

showAuthView();

showAlert(‘Logged out successfully.’, ‘success’);

}

/\*\*

\* Show alert (global) inside container at top

\* @param {string} message

\* @param {‘error’|’success’} type

\*/

Function showAlert(message, type=’success’) {

// Create alert if missing

Let alertContainer = container.querySelector(‘.alert-global’);

If (!alertContainer) {

alertContainer = document.createElement(‘div’);

alertContainer.className = ‘alert alert-global’;

alertContainer.style.cssText = `

position: fixed;

top: 15px;

left: 50%;

transform: translateX(-50%);

z-index: 9999;

max-width: 350px;

width: 90%;

text-align: center;

font-weight: 600;

font-size: 1rem;

border-radius: 10px;

padding: 12px;

`;

Container.parentElement.appendChild(alertContainer);

}

alertContainer.textContent = message;

alertContainer.classList.remove(‘alert-error’, ‘alert-success’);

alertContainer.classList.add(type === ‘error’ ? ‘alert-error’ : ‘alert-success’);

alertContainer.style.display = ‘block’;

setTimeout(() => {

alertContainer.style.display = ‘none’;

}, 3500);

}

// On page load, check if logged in

Const loggedInEmail = getLoggedInUser();

If (loggedInEmail) {

// Load users to get username

Let username = ‘’;

Try {

Const users = JSON.parse(localStorage.getItem(‘users-db’) || ‘{}’);

If (users[loggedInEmail]) {

Username = users[loggedInEmail].username || ‘’;

}

} catch {}

showPostView(loggedInEmail, username);

showAlert(`Welcome back, ${username || loggedInEmail}`, ‘success’);

} else {

showAuthView();

}

</content>

</create\_file>

<!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8” />

<meta name=”viewport” content=”width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no” />

<title>Login / Register / Post</title>

<style>

/\* Reset and base \*/

* {

Box-sizing: border-box;

}

Body, html {

Margin: 0; padding: 0; height: 100%;

Font-family: ‘Segoe UI’, Tahoma, Geneva, Verdana, sans-serif;

Background: linear-gradient(135deg, #667eea, #764ba2);

Color: #333;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

Display: flex;

Justify-content: center;

Align-items: center;

Min-height: 600px;

Min-width: 350px;

Padding: 20px;

}

.container {

Max-width: 350px;

Width: 100%;

Max-height: 600px;

Background: rgba(255,255,255,0.15);

Border-radius: 15px;

Box-shadow: 0 10px 30px rgba(0,0,0,0.2);

Backdrop-filter: blur(12px);

Color: #fff;

Padding: 25px;

Display: flex;

Flex-direction: column;

Box-sizing: border-box;

}

H2 {

Color: #fff;

Text-shadow: 0 0 5px rgba(0,0,0,0.4);

Font-weight: 700;

Margin-bottom: 20px;

Text-align: center;

}

.alert {

Padding: 12px;

Margin-bottom: 14px;

Border-radius: 8px;

Font-weight: 600;

Font-size: 0.9rem;

Text-align: center;

}

.alert-success {

Background: #7dd57d;

Color: #1f3c17;

Box-shadow: 0 0 12px #6bc87b88;

}

.alert-error {

Background: #e56a6a;

Color: #5a0f0f;

Box-shadow: 0 0 12px #e56a6a88;

}

/\* Responsive for small screens \*/

@media (max-width: 400px) {

Body, html {

Padding: 10px;

Min-width: auto;

}

.container {

Max-width: 100%;

Max-height: none;

Min-height: 600px;

Border-radius: 10px;

Padding: 20px;

}

}

</style>

</head>

<body>

<div class=”container” id=”app-container”>

<!—Dynamic content will be rendered here 🡪

</div>

<script type=”module” src=”./main.js”></script>

</body>

</html>

</content>

</create\_file>

<create\_file>

<path>auth.js</path>

<content>

// auth.js

// Handles login and registration UI, validation, storage, alert messages

Const LS\_KEY\_USERS = ‘users-db’;

Const LS\_KEY\_LOGGED\_IN = ‘logged-in-user’;

/\*\*

\* SHA-256 hashing utility, returns hex string

\* @param {string} message

\* @returns {Promise<string>}

\*/

Async function sha256(message) {

Const msgBuffer = new TextEncoder().encode(message);

Const hashBuffer = await crypto.subtle.digest(‘SHA-256’, msgBuffer);

Const hashArray = Array.from(new Uint8Array(hashBuffer));

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

}

/\*\*

\* Load users object from localStorage

\* @returns {Object} users keyed by email

\*/

Function loadUsers() {

Try {

Return JSON.parse(localStorage.getItem(LS\_KEY\_USERS)) || {};

} catch {

Return {};

}

}

/\*\*

\* Save users object to localStorage

\* @param {Object} users

\*/

Function saveUsers(users) {

localStorage.setItem(LS\_KEY\_USERS, JSON.stringify(users));

}

/\*\*

\* Get currently logged-in user email

\* @returns {string|null}

\*/

Function getLoggedInUser() {

Return localStorage.getItem(LS\_KEY\_LOGGED\_IN);

}

/\*\*

\* Set logged-in user email

\* @param {string} email

\*/

Function setLoggedInUser(email) {

localStorage.setItem(LS\_KEY\_LOGGED\_IN, email);

}

/\*\*

\* Remove logged-in user (logout)

\*/

Function clearLoggedInUser() {

localStorage.removeItem(LS\_KEY\_LOGGED\_IN);

}

/\*\*

\* Creates the login/register form HTML string

\* @param {boolean} isLoginMode

\* @returns {string}

\*/

Function renderForm(isLoginMode = true) {

Return `

<h2>${isLoginMode ? ‘Login’ : ‘Register’}</h2>

<div id=”alert-container”></div>

<form id=”auth-form” autocomplete=”off” novalidate>

<div class=”form-group” style=”${isLoginMode ? ‘display:none;’ : ‘display:block;’ }”>

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

<input type=”text” id=”reg-username” name=”reg-username” autocomplete=”username” placeholder=”Enter username” ${isLoginMode ? ‘’ : ‘required’} />

</div>

<div class=”form-group”>

<label for=”auth-email”>Email</label>

<input type=”email” id=”auth-email” name=”auth-email” autocomplete=”email” placeholder=”Enter email” required />

</div>

<div class=”form-group”>

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

<input type=”password” id=”auth-password” name=”auth-password” autocomplete=”${isLoginMode ? ‘current-password’ : ‘new-password’}” placeholder=”Enter password” required />

</div>

<button type=”submit”>${isLoginMode ? ‘Login’ : ‘Register’}</button>

</form>

<div class=”toggle-link” id=”toggle-auth”>${isLoginMode ? “Don’t have an account? Register” : “Already have an account? Login”}</div>

`;

}

/\*\*

\* Shows an alert message inside container

\* @param {HTMLElement} container

\* @param {string} message

\* @param {‘success’|’error’} type

\*/

Function showAlert(container, message, type=’success’) {

Container.innerHTML = ‘’;

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

div.className = ‘alert ‘ + (type===’error’?’alert-error’:’alert-success’);

div.textContent = message;

container.appendChild(div);

// Remove after 3.5 seconds

setTimeout(() => {

if(container.contains(div)) container.removeChild(div);

}, 3500);

}

/\*\*

\* Initializes the auth UI and logic inside given container

\* @param {HTMLElement} container

\* @param {function} onAuthSuccess Callback(email, username)

\*/

Function initAuth(container, onAuthSuccess) {

// Initial mode login

Let isLoginMode = true;

Function render() {

Container.innerHTML = renderForm(isLoginMode);

// Elements

Const authForm = container.querySelector(‘#auth-form’);

Const toggleLink = container.querySelector(‘#toggle-auth’);

Const alertContainer = container.querySelector(‘#alert-container’);

Const regUsernameInput = container.querySelector(‘#reg-username’);

Const emailInput = container.querySelector(‘#auth-email’);

Const passwordInput = container.querySelector(‘#auth-password’);

Const submitBtn = authForm.querySelector(‘button[type=”submit”]’);

toggleLink.onclick = () => {

isLoginMode = !isLoginMode;

render();

};

authForm.onsubmit = async € => {

e.preventDefault();

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

const password = passwordInput.value.trim();

if (isLoginMode) {

// Login

If (!email || !password) {

showAlert(alertContainer, ‘Please fill all required fields.’, ‘error’);

return;

}

Const users = loadUsers();

If (!users[email]) {

showAlert(alertContainer, ‘No account with that email. Please register.’, ‘error’);

return;

}

Const user = users[email];

Const passHash = await sha256(password);

If (passHash !== user.passwordHash) {

showAlert(alertContainer, ‘Incorrect password. Try again.’, ‘error’);

return;

}

setLoggedInUser(email);

showAlert(alertContainer, ‘Login successful! Redirecting…’, ‘success’);

setTimeout(() => onAuthSuccess(email, user.username), 1200);

} else {

// Register

Const username = regUsernameInput.value.trim();

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

showAlert(alertContainer, ‘Please fill all required fields.’, ‘error’);

return;

}

If (username.length < 3) {

showAlert(alertContainer, ‘Username must be at least 3 characters.’, ‘error’);

return;

}

Const users = loadUsers();

If (users[email]) {

showAlert(alertContainer, ‘Email already registered. Please login.’, ‘error’);

return;

}

Const passwordHash = await sha256(password);

Users[email] = { username, email, passwordHash };

saveUsers(users);

setLoggedInUser(email);

showAlert(alertContainer, ‘Registration successful! Redirecting…’, ‘success’);

setTimeout(() => onAuthSuccess(email, username), 1200);

}

};

}

Render();

Return {

toggleMode() {

isLoginMode = !isLoginMode;

render();

},

clearSession() {

clearLoggedInUser();

},

getLoggedInUser() {

return getLoggedInUser();

}

};

}

Export { initAuth, getLoggedInUser, clearLoggedInUser };

</content>

</create\_file>

<create\_file>

<path>post.js</path>

<content>

// post.js

// Handles posting UI, hashing post content, and output

/\*\*

\* SHA-256 hashing utility, returns hex string

\* @param {string} message

\* @returns {Promise<string>}

\*/

Async function sha256(message) {

Const msgBuffer = new TextEncoder().encode(message);

Const hashBuffer = await crypto.subtle.digest(‘SHA-256’, msgBuffer);

Const hashArray = Array.from(new Uint8Array(hashBuffer));

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

}

/\*\*

\* Creates HTML for the posting form

\* @param {string} username Optional username to greet user

\* @returns {string}

\*/

Function renderPostingPage(username=’’) {

Const greet = username ? `<p style=”margin-bottom:15px; font-weight:600; color:#cfcce0; font-size:1rem; text-align:center;”>Welcome <strong>${escapeHtml(username)}</strong>! Create a new post below.</p>` : ‘’;

Return `

<h2>New Post</h2>

${greet}

<div id=”alert-container”></div>

<form id=”post-form” autocomplete=”off” novalidate>

<div class=”form-group”>

<label for=”post-title”>Post Title</label>

<input type=”text” id=”post-title” name=”post-title” placeholder=”Enter title” required />

</div>

<div class=”form-group”>

<label for=”post-content”>Post Content</label>

<textarea id=”post-content” name=”post-content” placeholder=”Write your content here…” required></textarea>

</div>

<button type=”submit”>Submit Post</button>

</form>

<pre id=”post-hash-output” style=”display:none; background:#2f2d38; color:#a6a6e9; padding:12px; border-radius:10px; font-family: monospace; font-size: 0.9rem; user-select: all; white-space: pre-wrap;”></pre>

<button id=”logout-btn” style=”margin-top:20px; background:#bb4949; border:none; border-radius:10px; padding:10px; font-weight:bold; width: 100%; cursor:pointer;”>Logout</button>

`;

}

/\*\*

\* Escape HTML entities to prevent injection

\* @param {string} unsafe

\* @returns {string}

\*/

Function escapeHtml(unsafe) {

Return unsafe.replace(/[&<”’>]/g, m => ({

‘&’: ‘&amp;’,

‘<’: ‘&lt;’,

‘>’: ‘&gt;’,

‘”’: ‘&quot;’,

“’”: ‘&#39;’

})[m]);

}

/\*\*

\* Shows alert message inside given container

\* @param {HTMLElement} container

\* @param {string} message

\* @param {‘success’|’error’} type

\*/

Function showAlert(container, message, type=’success’) {

Container.innerHTML = ‘’;

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

div.className = ‘alert ‘ + (type===’error’?’alert-error’:’alert-success’);

div.textContent = message;

container.appendChild(div);

// Auto remove after 3.5 sec

setTimeout(() => {

if(container.contains(div)) container.removeChild(div);

}, 3500);

}

/\*\*

\* Initialize the posting page inside given container

\* @param {HTMLElement} container

\* @param {string} username

\* @param {function} onLogout Callback on logout

\*/

Function initPost(container, username, onLogout) {

Container.innerHTML = renderPostingPage(username);

Const postForm = container.querySelector(‘#post-form’);

Const hashOutput = container.querySelector(‘#post-hash-output’);

Const alertContainer = container.querySelector(‘#alert-container’);

Const logoutBtn = container.querySelector(‘#logout-btn’);

postForm.onsubmit = async € => {

e.preventDefault();

const title = postForm.elements[‘post-title’].value.trim();

const content = postForm.elements[‘post-content’].value.trim();

if (!title || !content) {

showAlert(alertContainer, ‘Please enter both post title and content.’, ‘error’);

return;

}

Const toHash = `Title: ${title}\nContent: ${content}`;

Const hash = await sha256(toHash);

hashOutput.textContent = `SHA-256 Hash of your post:\n${hash}`;

hashOutput.style.display = ‘block’;

showAlert(alertContainer, ‘Post submitted and hashed!’, ‘success’);

postForm.reset();

};

logoutBtn.onclick = () => {

if (typeof onLogout === ‘function’) {

onLogout();

}

};

}

Export { initPost };

</content>

</create\_file>

<create\_file>

<path>main.js</path>

<content>

// main.js – App entry point and controller

Import { initAuth, clearLoggedInUser, getLoggedInUser } from ‘./auth.js’;

Import { initPost } from ‘./post.js’;

Const container = document.getElementById(‘app-container’);

/\*\*

\* Shows the login/register view

\*/

Function showAuthView() {

// Initialize the auth module inside container

initAuth(container, (email, username) => {

showPostView(email, username);

});

}

/\*\*

\* Shows posting form view with user greeted

\* @param {string} email

\* @param {string} username

\*/

Function showPostView(email, username) {

initPost(container, username || email, () => {

logout();

});

}

/\*\*

\* Logout the user and show auth view

\*/

Function logout() {

clearLoggedInUser();

showAuthView();

showAlert(‘Logged out successfully.’, ‘success’);

}

/\*\*

\* Show alert (global) inside container at top

\* @param {string} message

\* @param {‘error’|’success’} type

\*/

Function showAlert(message, type=’success’) {

// Create alert if missing

Let alertContainer = container.querySelector(‘.alert-global’);

If (!alertContainer) {

alertContainer = document.createElement(‘div’);

alertContainer.className = ‘alert alert-global’;

alertContainer.style.cssText = `

position: fixed;

top: 15px;

left: 50%;

transform: translateX(-50%);

z-index: 9999;

max-width: 350px;

width: 90%;

text-align: center;

font-weight: 600;

font-size: 1rem;

border-radius: 10px;

padding: 12px;

`;

Container.parentElement.appendChild(alertContainer);

}

alertContainer.textContent = message;

alertContainer.classList.remove(‘alert-error’, ‘alert-success’);

alertContainer.classList.add(type === ‘error’ ? ‘alert-error’ : ‘alert-success’);

alertContainer.style.display = ‘block’;

setTimeout(() => {

alertContainer.style.display = ‘none’;

}, 3500);

}

// On page load, check if logged in

Const loggedInEmail = getLoggedInUser();

If (loggedInEmail) {

// Load users to get username

Let username = ‘’;

Try {

Const users = JSON.parse(localStorage.getItem(‘users-db’) || ‘{}’);

If (users[loggedInEmail]) {

Username = users[loggedInEmail].username || ‘’;

}

} catch {}

showPostView(loggedInEmail, username);

showAlert(`Welcome back, ${username || loggedInEmail}`, ‘success’);

} else {

showAuthView();

}

</content>

</create\_file>