// Ensure these are defined at the top level and exposed globally

window.showTab = function(tabId) {

const tabs = document.querySelectorAll('.tab-content');

const buttons = document.querySelectorAll('.tab-btn');

tabs.forEach(tab => tab.classList.remove('active'));

buttons.forEach(btn => btn.classList.remove('active'));

document.getElementById(tabId + '-tab').classList.add('active');

document.querySelector(`.tab-btn[onclick="showTab('${tabId}')"]`).classList.add('active');

if (tabId === 'dashboard') {

showDashboard('mockUserId123');

} else if (tabId === 'people') {

showAvailablePersons();

}

};

window.showAvailablePersons = function(radius = 5000) {

const preferredName = localStorage.getItem('preferredName') || 'Anonymous';

const peopleList = document.getElementById('people-list');

const connectPersonBtn = document.getElementById('connect-person-btn');

const loadingSpinner = document.getElementById('loading-spinner');

if (!peopleList || !connectPersonBtn || !loadingSpinner) {

console.error('Required DOM elements not found:', {

peopleList: !!peopleList,

connectPersonBtn: !!connectPersonBtn,

loadingSpinner: !!loadingSpinner

});

return;

}

loadingSpinner.classList.remove('hidden');

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(async position => {

const lat = position.coords.latitude;

const lon = position.coords.longitude;

try {

const response = await fetch(`https://chatroulette-lite.onrender.com/api/users/nearby?latitude=${lat}&longitude=${lon}&radius=${radius}`);

if (!response.ok) throw new Error(`HTTP error! Status: ${response.status}`);

const nearbyUsers = await response.json();

console.log('Nearby users response:', nearbyUsers);

peopleList.innerHTML = '<h3>Available Persons Nearby</h3>';

if (nearbyUsers.length === 0) {

peopleList.innerHTML += '<p>No nearby users found. Try adjusting the radius.</p>';

} else {

nearbyUsers.forEach(user => {

const prefs = user.preferences || {};

const userName = user.preferredName || 'Anonymous';

const ageRange = prefs.ageRange ? `${prefs.ageRange.min}-${prefs.ageRange.max}` : 'Not specified';

peopleList.innerHTML += `

<p>

<div class="user-details">

<span class="name">${userName} (User ${user.\_id})</span>

<span class="info">${Math.round(user.dist.calculated / 1000)} km away (Topics: ${prefs.topics || 'None'}, Age Range: ${ageRange})</span>

</div>

<button class="connect-person-btn-small" onclick="connectToPerson('${user.\_id}')">Connect</button>

</p>`;

});

}

connectPersonBtn.classList.remove('hidden');

} catch (error) {

peopleList.innerHTML = '<p>Error fetching nearby users. Details: ' + error.message + '</p>';

} finally {

loadingSpinner.classList.add('hidden');

}

}, error => {

peopleList.innerHTML = '<p>Location access denied or error. Showing mock data.</p>';

const mockPersons = [

{ id: 1, name: preferredName, distance: '1.2 km away' },

{ id: 2, name: preferredName, distance: '2.5 km away' }

];

mockPersons.forEach(person => {

peopleList.innerHTML += `

<p>

<div class="user-details">

<span class="name">${person.name}</span>

<span class="info">${person.distance}</span>

</div>

<button class="connect-person-btn-small" onclick="connectToPerson('${person.id}')">Connect</button>

</p>`;

});

connectPersonBtn.classList.remove('hidden');

loadingSpinner.classList.add('hidden');

});

} else {

peopleList.innerHTML = '<p>Geolocation not supported. Showing mock data.</p>';

const mockPersons = [

{ id: 1, name: preferredName, distance: '1.2 km away' },

{ id: 2, name: preferredName, distance: '2.5 km away' }

];

mockPersons.forEach(person => {

peopleList.innerHTML += `

<p>

<div class="user-details">

<span class="name">${person.name}</span>

<span class="info">${person.distance}</span>

</div>

<button class="connect-person-btn-small" onclick="connectToPerson('${person.id}')">Connect</button>

</p>`;

});

connectPersonBtn.classList.remove('hidden');

loadingSpinner.classList.add('hidden');

}

};

window.connectToPerson = function(userId) {

const preferredName = localStorage.getItem('preferredName') || 'Anonymous';

console.log('Initiating connection to user:', userId, 'with preferredName:', preferredName);

fetch(`https://chatroulette-lite.onrender.com/api/connect/${userId}`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ userId, preferredName })

})

.then(response => {

console.log('Fetch response status:', response.status);

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

return response.json();

})

.then(data => {

console.log('Fetch response data:', data);

if (data.success) {

document.getElementById('status').textContent = `Connected with ${data.otherUserName || 'Anonymous'} (User ${userId})`;

document.getElementById('topic').classList.remove('hidden');

document.getElementById('input-container').classList.remove('hidden');

document.getElementById('connect-btn').classList.add('hidden');

startTimer();

setupPusher(preferredName, userId);

} else {

console.error('Connection failed:', data.error);

alert('Failed to connect: ' + (data.error || 'Unknown error'));

}

})

.catch(error => {

console.error('Error connecting to user:', error);

alert('Error connecting: ' + error.message);

});

};

// Rest of the script.js content remains the same, wrapped in document.addEventListener('DOMContentLoaded', () => { ... })

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

// Generate a unique client ID for this tab

const clientId = Math.random().toString(36).substring(2, 15);

// Pusher setup

const pusher = new Pusher('b14541edb68153cc4354', {

cluster: 'ap2',

authEndpoint: 'https://chatroulette-lite.onrender.com/pusher/auth',

auth: {

params: {

user\_id: clientId,

preferredName: localStorage.getItem('preferredName') || 'Anonymous'

}

}

});

const channel = pusher.subscribe('chat-channel');

// DOM elements

const connectBtn = document.getElementById('connect-btn');

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

const chatBox = document.getElementById('chat-box');

const messageInput = document.getElementById('message-input');

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

const topicDiv = document.getElementById('topic');

const topicText = document.getElementById('topic-text');

const timerDiv = document.getElementById('timer');

const peopleList = document.getElementById('people-list');

const connectPersonBtn = document.getElementById('connect-person-btn');

const consentModal = document.getElementById('consent-modal');

const consentYes = document.getElementById('consent-yes');

const consentNo = document.getElementById('consent-no');

const preferencesCard = document.getElementById('preferences-card');

const preferencesModal = document.getElementById('preferences-modal');

const preferenceSelect = document.getElementById('preference-select');

const preferenceSave = document.getElementById('preference-save');

const preferenceCancel = document.getElementById('preference-cancel');

const preferencesInfo = document.getElementById('preferences-info');

console.log('Consent Modal:', consentModal);

console.log('Consent Yes:', consentYes);

console.log('Consent No:', consentNo);

if (!consentModal || !consentYes || !consentNo) {

console.error('One or more consent modal elements not found in DOM');

return;

}

// Consent modal logic

const locationConsent = localStorage.getItem('locationConsent');

console.log('Location Consent from localStorage:', locationConsent);

if (locationConsent !== 'true') {

consentModal.style.display = 'block';

console.log('Consent modal displayed');

} else {

showAvailablePersons();

}

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

console.log('Yes button clicked');

consentModal.style.display = 'none';

localStorage.setItem('locationConsent', 'true');

showAvailablePersons();

});

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

console.log('No button clicked');

consentModal.style.display = 'none';

localStorage.setItem('locationConsent', 'false');

peopleList.innerHTML = '<p>Location sharing declined. Using mock data.</p>';

const mockPersons = [

{ id: 1, name: 'Alex', distance: '1.2 km away' },

{ id: 2, name: 'Sam', distance: '2.5 km away' }

];

mockPersons.forEach(person => {

peopleList.innerHTML += `<p>${person.name} - ${person.distance} <button class="connect-person-btn-small" onclick="connectToPerson('${person.id}')">Connect</button></p>`;

});

connectPersonBtn.classList.remove('hidden');

});

if (preferencesCard && preferencesModal && preferenceSelect && preferenceSave && preferenceCancel && preferencesInfo) {

console.log('All dashboard elements found, attaching event listeners...');

preferencesModal.classList.add('hidden');

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

preferencesModal.classList.remove('hidden');

preferenceSelect.value = '';

});

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

const selectedPreference = preferenceSelect.value;

if (selectedPreference) {

preferencesInfo.textContent = `Pref: ${selectedPreference}`;

preferencesInfo.classList.add('green-text');

preferencesModal.classList.add('hidden');

fetch('https://chatroulette-lite.onrender.com/api/user/update-preferences', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ userId: 'mockUserId123', preferences: { topics: [selectedPreference] } })

})

.then(response => response.json())

.then(data => console.log('Preference saved:', data))

.catch(error => console.error('Error saving preference:', error));

}

});

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

preferencesModal.classList.add('hidden');

});

} else {

console.error('One or more dashboard elements not found in DOM');

}

function sendTalkRequest(name) {

console.log(`Talk request sent to ${name}`);

alert(`Talk request sent to ${name}!`);

}

let isConnected = false;

let timeLeft = 300;

let personId = null;

let lastMessageId = '';

let partnerName = '';

const topics = [

'What’s your dream vacation spot?',

'Talk about your favorite movie',

'What’s the best thing you’ve eaten recently?',

'Share a fun fact you know'

];

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

if (!isConnected) {

showTab('people');

} else {

resetChat();

}

});

sendBtn.addEventListener('click', sendMessage);

messageInput.addEventListener('keypress', (e) => {

if (e.key === 'Enter' && messageInput.value.trim() !== '') {

sendMessage();

}

});

function setupPusher(preferredName, userId) {

const presenceChannel = pusher.subscribe(`presence-chat-${userId}`);

presenceChannel.bind('message', data => {

const chatBox = document.getElementById('chat-box');

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

messageDiv.classList.add('message', data.sender === preferredName ? 'sent' : 'received');

messageDiv.innerHTML = `${data.sender}: ${data.message}<span class="time">${new Date().toLocaleTimeString()}</span>`;

chatBox.appendChild(messageDiv);

chatBox.scrollTop = chatBox.scrollHeight;

});

document.getElementById('send-btn').addEventListener('click', () => {

const messageInput = document.getElementById('message-input');

const message = messageInput.value.trim();

if (message) {

presenceChannel.trigger('client-message', {

message: message,

sender: preferredName

});

messageInput.value = '';

}

});

}

function sendMessage() {

const message = messageInput.value.trim();

if (message && isConnected) {

const now = new Date().toLocaleTimeString();

const messageId = `${message}-${now}`;

chatBox.innerHTML += `<div class="message sent"><span>${message}</span><span class="time">${now}</span></div>`;

chatBox.scrollTop = chatBox.scrollHeight;

fetch('https://chatroulette-lite.onrender.com/message', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ text: message, from: clientId, to: personId, messageId })

}).catch(err => console.error('Error sending message:', err));

messageInput.value = '';

lastMessageId = messageId;

}

}

function startTimer() {

const timer = setInterval(() => {

timeLeft--;

const minutes = Math.floor(timeLeft / 60);

const seconds = timeLeft % 60;

timerDiv.textContent = `${minutes}:${seconds < 10 ? '0' + seconds : seconds}`;

if (timeLeft <= 0) {

clearInterval(timer);

resetChat();

}

}, 1000);

}

function resetChat() {

isConnected = false;

personId = null;

partnerName = '';

connectBtn.textContent = 'Connect';

status.textContent = 'Click "Connect" to start chatting!';

topicDiv.classList.add('hidden');

document.getElementById('input-container').classList.add('hidden');

chatBox.innerHTML = '';

timeLeft = 300;

lastMessageId = '';

}

function showDashboard(userId) {

const preferredName = localStorage.getItem('preferredName') || 'Anonymous';

const userInfoElement = document.getElementById('user-info');

if (!userInfoElement) {

console.error('User info element not found');

return;

}

userInfoElement.textContent = `Welcome, ${preferredName}!`;

fetch(`https://chatroulette-lite.onrender.com/api/user/dashboard?userId=${userId}`)

.then(response => response.json())

.then(data => {

if (navigator.geolocation) {

navigator.geolocation.getCurrentPosition(

() => document.getElementById('location-info').textContent = 'Shared',

() => document.getElementById('location-info').textContent = 'Not shared'

);

} else {

document.getElementById('location-info').textContent = 'Not supported';

}

document.getElementById('online-status').textContent = 'Online';

document.getElementById('online-status').classList.add('green-text');

document.getElementById('preferences-info').textContent = data.preferences ? `Pref: ${data.preferences.topics?.join(', ')}` : 'Not defined';

document.getElementById('last-chatted').textContent = new Date().toLocaleString();

})

.catch(error => console.error('Error loading dashboard:', error));

const historyLog = document.getElementById('history-log');

const mockHistory = [

{ name: preferredName, online: true, age: 22 },

{ name: 'Alex', online: false, age: 25 },

{ name: 'Morgan', online: true, age: 30 }

];

historyLog.innerHTML = '';

mockHistory.forEach(person => {

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

entry.classList.add('history-entry');

entry.innerHTML = `

<div class="history-details">

<div class="name">${person.name}</div>

<div class="info">Status: <span class="${person.online ? 'status-online' : 'status-offline'}">${person.online ? 'Online' : 'Offline'}</span> | Age: ${person.age}</div>

</div>

<button class="talk-request-btn" onclick="sendTalkRequest('${person.name}')">Talk Request</button>

`;

historyLog.appendChild(entry);

});

}

channel.bind('message', (data) => {

console.log('Received message event:', data);

const messageId = data.messageId || `${data.text}-${data.from}`;

if (isConnected && data.to === personId && data.from !== clientId && messageId !== lastMessageId) {

const now = new Date().toLocaleTimeString();

if (data.text.includes('Connected with')) {

chatBox.innerHTML += `<div class="message received"><span>${data.text}</span><span class="time">${now}</span></div>`;

} else {

chatBox.innerHTML += `<div class="message received"><span>${data.text}</span><span class="time">${now}</span></div>`;

}

chatBox.scrollTop = chatBox.scrollHeight;

lastMessageId = messageId;

}

});

// Initialize with chat tab active

showTab('chat');

// Consent modal

consentModal.style.display = 'block';

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

console.log('Yes button clicked');

consentModal.style.display = 'none';

localStorage.setItem('locationConsent', 'true');

showAvailablePersons();

});

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

console.log('No button clicked');

consentModal.style.display = 'none';

localStorage.setItem('locationConsent', 'false');

peopleList.innerHTML = '<p>Location sharing declined. Using mock data.</p>';

const mockPersons = [

{ id: 1, name: 'Alex', distance: '1.2 km away' },

{ id: 2, name: 'Sam', distance: '2.5 km away' }

];

mockPersons.forEach(person => {

peopleList.innerHTML += `<p>${person.name} - ${person.distance} <button class="connect-person-btn-small" onclick="connectToPerson('${person.id}')">Connect</button></p>`;

});

connectPersonBtn.classList.remove('hidden');

});

if (localStorage.getItem('locationConsent') === 'true') {

consentModal.style.display = 'none';

showAvailablePersons();

}

});