// --- SECTION 1: FIREBASE CONFIGURATION & INITIALIZATION ---

// Your web app's Firebase configuration (using the provided keys)

const firebaseConfig = {

apiKey: "AIzaSyBhyOdPB98l0SBgUduxunjx5s3HJBgWyQM",

authDomain: "classroompictionaryapp.firebaseapp.com",

projectId: "classroompictionaryapp",

storageBucket: "classroompictionaryapp.firebasestorage.app",

messagingSenderId: "812924722367",

appId: "1:812924722367:web:05fa8f8f4ca15c8fb13a0a",

measurementId: "G-7JLV3J95V7"

};

// Initialize Firebase

const app = firebase.initializeApp(firebaseConfig);

const db = app.firestore();

const analytics = firebase.analytics(app);

// Firestore Collection/Document References

const GAME\_SESSION\_DOC = db.collection('gameSession').doc('activeSession');

const USERS\_COLLECTION = db.collection('users');

const ATTEMPTS\_COLLECTION = db.collection('attempts');

const ULTIMATE\_LEADERBOARD\_COLLECTION = db.collection('ultimateLeaderboard');

const DRAWING\_DOC = db.collection('drawingData').doc('currentDrawing');

// --- SECTION 2: GLOBAL STATE & UI ELEMENTS ---

let userID = localStorage.getItem('pictionaryUserID') || `guest-${Math.random().toString(36).substring(2, 9)}`;

let username = '';

let isDrawer = false;

let gameStatus = 'LOBBY';

let currentPromptWord = '';

let currentDrawerID = '';

let activeUsers = {}; // Cache of users for quick lookup

let timerInterval = null;

const TIMER\_DURATION = 120; // 120 seconds

// UI Elements

const ui = {

lobbyScreen: document.getElementById('lobby-screen'),

gameScreen: document.getElementById('game-screen'),

gameEndScreen: document.getElementById('game-end-screen'),

usernameInput: document.getElementById('username-input'),

joinGameBtn: document.getElementById('join-game-btn'),

teacherControls: document.getElementById('teacher-controls'),

promptListInput: document.getElementById('prompt-list-input'),

startGameBtn: document.getElementById('start-game-btn'),

endGameBtn: document.getElementById('end-game-btn'),

leaderboardList: document.getElementById('leaderboard-list'),

currentPromptWordDisplay: document.getElementById('current-prompt-word'),

promptDisplay: document.getElementById('prompt-display'),

gameTimer: document.getElementById('game-timer'),

drawingTools: document.getElementById('drawing-tools'),

drawingCanvas: document.getElementById('drawing-canvas'),

attemptsList: document.getElementById('attempts-list'),

guessInput: document.getElementById('guess-input'),

submitGuessBtn: document.getElementById('submit-guess-btn'),

finalScores: document.getElementById('final-scores'),

returnToLobbyBtn: document.getElementById('return-to-lobby-btn'),

messageOverlay: document.getElementById('message-overlay'),

overlayText: document.getElementById('overlay-text')

};

// --- SECTION 3: CORE GAME LOGIC & FUNCTIONS ---

// 4. SCORING FUNCTION

function calculatePoints(timeRemaining) {

if (timeRemaining < 0) timeRemaining = 0;

// Formula: Points = ROUNDUP(100 - ((100 - 10) / 120) \* (120 - TimeRemaining))

const points = Math.ceil(100 - ((90 / TIMER\_DURATION) \* (TIMER\_DURATION - timeRemaining)));

return Math.max(10, Math.min(100, points)); // Ensure points are between 10 and 100

}

// 4. TIMER LOGIC

function startTimer(startTime) {

if (timerInterval) clearInterval(timerInterval);

timerInterval = setInterval(() => {

const timeElapsed = Math.floor((Date.now() - startTime.toDate().getTime()) / 1000);

const timeRemaining = TIMER\_DURATION - timeElapsed;

const minutes = String(Math.floor(timeRemaining / 60)).padStart(2, '0');

const seconds = String(timeRemaining % 60).padStart(2, '0');

const currentPoints = calculatePoints(timeRemaining);

ui.gameTimer.textContent = `${minutes}:${seconds} (Points: ${currentPoints})`;

if (timeRemaining <= 0) {

clearInterval(timerInterval);

handleRoundTimeout();

}

}, 1000);

}

// 4. ROUND END (TIMEOUT)

async function handleRoundTimeout() {

console.log("Round timed out. No points awarded.");

showOverlay("Time's Up! No one guessed it. 😔", 3000);

// Trigger new round start via gameSession update

await startNewRound();

}

// 4. ROUND END (VICTORY)

async function handleCorrectGuess(guesserID) {

clearInterval(timerInterval);

const gameSessionSnapshot = await GAME\_SESSION\_DOC.get();

const gameSessionData = gameSessionSnapshot.data();

const startTime = gameSessionData.startTime.toDate().getTime();

const timeElapsed = Math.floor((Date.now() - startTime) / 1000);

const timeRemaining = TIMER\_DURATION - timeElapsed;

const pointsAwarded = calculatePoints(timeRemaining);

// 4. Update Guesser's score

const guesserRef = USERS\_COLLECTION.doc(guesserID);

await db.runTransaction(async (transaction) => {

const guesserDoc = await transaction.get(guesserRef);

if (guesserDoc.exists) {

const currentPoints = guesserDoc.data().currentPoints || 0;

transaction.update(guesserRef, { currentPoints: currentPoints + pointsAwarded });

}

});

// 4. Update Drawer's score

const drawerRef = USERS\_COLLECTION.doc(gameSessionData.drawerID);

await db.runTransaction(async (transaction) => {

const drawerDoc = await transaction.get(drawerRef);

if (drawerDoc.exists) {

const currentPoints = drawerDoc.data().currentPoints || 0;

transaction.update(drawerRef, { currentPoints: currentPoints + pointsAwarded });

}

});

showOverlay(`Correct! ${activeUsers[guesserID].username} guessed the word! (+${pointsAwarded} points) 🎉`, 3000);

// Start a new round

await startNewRound(gameSessionData.roundNumber + 1);

}

// 4. START NEW ROUND

async function startNewRound(nextRound = 1) {

const usersSnapshot = await USERS\_COLLECTION.get();

const activeUserIDs = usersSnapshot.docs.map(doc => doc.id);

if (activeUserIDs.length === 0) {

console.error("No active users to start a round.");

return;

}

// Determine the next drawer (simple round-robin or skip the previous drawer)

// For simplicity, we just choose a random user for now.

const nextDrawerID = activeUserIDs[Math.floor(Math.random() \* activeUserIDs.length)];

const gameSessionSnapshot = await GAME\_SESSION\_DOC.get();

const promptList = (gameSessionSnapshot.exists && gameSessionSnapshot.data().promptList) ? gameSessionSnapshot.data().promptList : ["APPLE", "TRAIN", "HOUSE"];

if (promptList.length === 0) {

console.error("Prompt list is empty. Cannot start round.");

// Consider a fallback or automatic Game End

return;

}

const nextPromptWord = promptList[Math.floor(Math.random() \* promptList.length)];

// Reset attempts and drawing data first (for smooth transition)

// Note: Deleting a whole collection is complex and slow in Firestore, so we just clear the content of the real-time drawing doc.

await DRAWING\_DOC.set({ lines: [] });

// Update the main game session

await GAME\_SESSION\_DOC.update({

status: 'IN\_PROGRESS',

roundNumber: nextRound,

drawerID: nextDrawerID,

promptWord: nextPromptWord.toUpperCase(),

startTime: firebase.firestore.FieldValue.serverTimestamp(),

});

// We rely on the gameSession listener to update the UI

}

// 4. GAME END (TEACHER ACTION)

async function endGame() {

clearInterval(timerInterval);

// 1. Get final scores and update lifetime score

const usersSnapshot = await USERS\_COLLECTION.get();

const finalScores = [];

await db.runTransaction(async (transaction) => {

for (const doc of usersSnapshot.docs) {

const userData = doc.data();

const currentPoints = userData.currentPoints || 0;

const finalUsername = userData.username;

if (currentPoints > 0) {

finalScores.push({ username: finalUsername, score: currentPoints });

// Update totalLifetimeScore

const ulRef = ULTIMATE\_LEADERBOARD\_COLLECTION.doc(doc.id);

const ulDoc = await transaction.get(ulRef);

if (ulDoc.exists) {

const currentTotal = ulDoc.data().totalLifetimeScore || 0;

transaction.update(ulRef, { totalLifetimeScore: currentTotal + currentPoints });

} else {

transaction.set(ulRef, { username: finalUsername, totalLifetimeScore: currentPoints });

}

}

// Reset currentPoints for the next session

transaction.update(doc.ref, { currentPoints: 0 });

}

});

// 2. Set game status to GAME\_END and display final scores

await GAME\_SESSION\_DOC.update({ status: 'GAME\_END', finalScores: finalScores });

// UI update handled by listener

}

// 3. UI OVERLAY

function showOverlay(text, duration) {

ui.overlayText.textContent = text;

ui.messageOverlay.classList.add('visible');

// Simple sound logic (not included due to vanilla JS constraints)

setTimeout(() => {

ui.messageOverlay.classList.remove('visible');

}, duration);

}

// --- SECTION 4: FIREBASE LISTENERS (REAL-TIME DATA) ---

// Listener 1: Game Session State

db.doc('gameSession/activeSession').onSnapshot(snapshot => {

if (!snapshot.exists) return;

const data = snapshot.data();

gameStatus = data.status;

currentDrawerID = data.drawerID;

currentPromptWord = data.promptWord;

// Determine current user's role

isDrawer = (data.status === 'IN\_PROGRESS' && currentDrawerID === userID);

// UI State Management

if (gameStatus === 'LOBBY') {

ui.lobbyScreen.classList.add('active');

ui.gameScreen.classList.remove('active');

ui.gameEndScreen.classList.remove('active');

if (userID === 'teacherID') ui.teacherControls.classList.remove('hidden'); // Simplified teacher check

clearInterval(timerInterval);

} else if (gameStatus === 'IN\_PROGRESS') {

ui.lobbyScreen.classList.remove('active');

ui.gameScreen.classList.add('active');

ui.gameEndScreen.classList.remove('active');

// Drawer/Guesser specific UI

ui.promptDisplay.classList.toggle('hidden', !isDrawer);

ui.drawingTools.classList.toggle('hidden', !isDrawer);

ui.guessInput.disabled = isDrawer;

ui.submitGuessBtn.disabled = isDrawer;

ui.currentPromptWordDisplay.textContent = isDrawer ? currentPromptWord : '...';

if (data.startTime) {

startTimer(data.startTime);

}

} else if (gameStatus === 'GAME\_END') {

ui.lobbyScreen.classList.remove('active');

ui.gameScreen.classList.remove('active');

ui.gameEndScreen.classList.add('active');

renderFinalScores(data.finalScores || []);

}

});

// Listener 2: Users and Live Leaderboard

db.collection('users').orderBy('currentPoints', 'desc').onSnapshot(snapshot => {

ui.leaderboardList.innerHTML = '';

activeUsers = {}; // Reset cache

snapshot.forEach(doc => {

const user = doc.data();

activeUsers[doc.id] = user;

const li = document.createElement('li');

li.textContent = user.username;

const scoreSpan = document.createElement('span');

scoreSpan.textContent = user.currentPoints || 0;

if (doc.id === currentDrawerID && gameStatus === 'IN\_PROGRESS') {

li.classList.add('is-drawer');

li.textContent += ' (Drawing)';

}

ui.leaderboardList.appendChild(li);

li.appendChild(scoreSpan);

});

});

// Listener 3: Attempts/Chat

db.collection('attempts').orderBy('timestamp', 'desc').limit(20).onSnapshot(snapshot => {

ui.attemptsList.innerHTML = '';

snapshot.forEach(doc => {

const attempt = doc.data();

const username = activeUsers[attempt.userID] ? activeUsers[attempt.userID].username : 'Unknown';

const li = document.createElement('li');

li.innerHTML = `<strong>${username}:</strong> ${attempt.guessText}`;

ui.attemptsList.appendChild(li);

});

});

// Listener 4: Drawing Data (Handles rendering the drawing)

db.doc('drawingData/currentDrawing').onSnapshot(snapshot => {

if (!snapshot.exists) return;

const data = snapshot.data();

if (data && data.lines) {

// Clear canvas and redraw

clearCanvas();

data.lines.forEach(line => {

drawLine(line.start, line.end, line.color, line.width, line.isEraser, false);

});

} else {

clearCanvas(); // If drawing data is empty

}

});

// --- SECTION 5: UI EVENT HANDLERS & INITIALIZATION ---

// 1. LOBBY/USER ENTRY

ui.joinGameBtn.addEventListener('click', async () => {

const inputUsername = ui.usernameInput.value.trim();

if (!inputUsername) {

alert('Please enter a username.');

return;

}

username = inputUsername;

localStorage.setItem('pictionaryUserID', userID); // Persist ID

// 4. User Entry: Add/Update user in Firestore

await USERS\_COLLECTION.doc(userID).set({

userID: userID,

username: username,

currentPoints: 0,

// totalLifetimeScore is managed in ultimateLeaderboard

}, { merge: true });

// Check if we need to initialize the game session

const sessionSnapshot = await GAME\_SESSION\_DOC.get();

if (!sessionSnapshot.exists) {

await GAME\_SESSION\_DOC.set({ status: 'LOBBY', roundNumber: 0, promptList: [] });

}

// SIMPLIFIED TEACHER LOGIC: The first person to join after a game reset is designated 'teacher' for controls

const usersSnapshot = await USERS\_COLLECTION.get();

if (usersSnapshot.docs.length === 1 && userID === usersSnapshot.docs[0].id) {

// Temporarily designate first user as teacher (not secure, for demonstration)

ui.teacherControls.classList.remove('hidden');

userID = 'teacherID'; // Temporary hack for teacher ID check

}

// Start listening for game state updates via the listener

});

// 2. TEACHER CONTROLS

ui.startGameBtn.addEventListener('click', async () => {

if (gameStatus !== 'LOBBY') return;

const prompts = ui.promptListInput.value.split('\n').map(p => p.trim()).filter(p => p.length > 0);

if (prompts.length === 0) {

alert("Please input a list of words/phrases.");

return;

}

// Update prompt list and start game (status will change to IN\_PROGRESS in startNewRound)

await GAME\_SESSION\_DOC.update({ promptList: prompts });

await startNewRound(1);

});

ui.endGameBtn.addEventListener('click', () => {

if (confirm("Are you sure you want to end the current game session?")) {

endGame();

}

});

// 3. GUESSING LOGIC

ui.submitGuessBtn.addEventListener('click', async () => {

const guessText = ui.guessInput.value.trim();

ui.guessInput.value = ''; // Clear input immediately

if (!guessText || gameStatus !== 'IN\_PROGRESS' || isDrawer) return;

// Add guess to attempts

await ATTEMPTS\_COLLECTION.add({

userID: userID,

guessText: guessText,

timestamp: firebase.firestore.FieldValue.serverTimestamp()

});

// 4. Correction Check

if (guessText.toUpperCase() === currentPromptWord.toUpperCase()) {

await handleCorrectGuess(userID);

}

});

// Helper for Game End Screen

function renderFinalScores(scores) {

ui.finalScores.innerHTML = '';

scores.sort((a, b) => b.score - a.score).forEach((item, index) => {

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

div.innerHTML = `<strong>#${index + 1} ${item.username}</strong><span>${item.score} Points</span>`;

ui.finalScores.appendChild(div);

});

}

ui.returnToLobbyBtn.addEventListener('click', async () => {

// Reset the game session to LOBBY status (all users' currentPoints were already reset in endGame)

await GAME\_SESSION\_DOC.update({ status: 'LOBBY', roundNumber: 0, drawerID: null, promptWord: null, startTime: null, finalScores: null });

// Reset attempts collection (simple clear not possible, rely on listener limit)

});

// --- SECTION 6: DRAWING CANVAS LOGIC ---

const ctx = ui.drawingCanvas.getContext('2d');

let isDrawing = false;

let currentTool = { color: '#000000', width: 5, isEraser: false };

let drawingCache = []; // Local cache of lines to be pushed to DB

// Canvas helper

function getCanvasCoords(event) {

const rect = ui.drawingCanvas.getBoundingClientRect();

const clientX = event.clientX || event.touches[0].clientX;

const clientY = event.clientY || event.touches[0].clientY;

return {

x: clientX - rect.left,

y: clientY - rect.top

};

}

// Drawing action - local render (true) or remote render (false)

function drawLine(start, end, color, width, isEraser, broadcast = true) {

ctx.beginPath();

ctx.strokeStyle = isEraser ? 'white' : color;

ctx.lineWidth = width;

ctx.lineCap = 'round';

ctx.lineJoin = 'round';

ctx.moveTo(start.x, start.y);

ctx.lineTo(end.x, end.y);

ctx.stroke();

if (broadcast && isDrawer) {

drawingCache.push({ start, end, color, width, isEraser });

// Throttle database writes (update once per stream)

if (drawingCache.length % 5 === 0) {

DRAWING\_DOC.update({ lines: firebase.firestore.FieldValue.arrayUnion(...drawingCache) })

.then(() => { drawingCache = []; }) // Clear cache on successful push

.catch(err => console.error("Error writing drawing data: ", err));

}

}

}

function clearCanvas() {

ctx.clearRect(0, 0, ui.drawingCanvas.width, ui.drawingCanvas.height);

}

// Set initial canvas background to white (essential for eraser)

ctx.fillStyle = "white";

ctx.fillRect(0, 0, ui.drawingCanvas.width, ui.drawingCanvas.height);

// --- Drawing Event Listeners ---

let lastPosition = { x: 0, y: 0 };

ui.drawingCanvas.addEventListener('mousedown', (e) => {

if (!isDrawer || gameStatus !== 'IN\_PROGRESS') return;

isDrawing = true;

lastPosition = getCanvasCoords(e);

});

ui.drawingCanvas.addEventListener('mousemove', (e) => {

if (!isDrawer || !isDrawing) return;

e.preventDefault(); // Prevent text selection/dragging on canvas

const newPosition = getCanvasCoords(e);

drawLine(lastPosition, newPosition, currentTool.color, currentTool.width, currentTool.isEraser, true);

lastPosition = newPosition;

});

ui.drawingCanvas.addEventListener('mouseup', () => {

if (!isDrawer) return;

isDrawing = false;

// Push any remaining cache items

if (drawingCache.length > 0) {

DRAWING\_DOC.update({ lines: firebase.firestore.FieldValue.arrayUnion(...drawingCache) })

.then(() => { drawingCache = []; })

.catch(err => console.error("Error writing drawing data: ", err));

}

});

ui.drawingCanvas.addEventListener('mouseout', () => {

if (!isDrawer) return;

isDrawing = false;

});

// Mobile/Touch Events

ui.drawingCanvas.addEventListener('touchstart', (e) => {

if (!isDrawer || gameStatus !== 'IN\_PROGRESS') return;

isDrawing = true;

lastPosition = getCanvasCoords(e);

e.preventDefault();

}, { passive: false });

ui.drawingCanvas.addEventListener('touchmove', (e) => {

if (!isDrawer || !isDrawing) return;

e.preventDefault();

const newPosition = getCanvasCoords(e);

drawLine(lastPosition, newPosition, currentTool.color, currentTool.width, currentTool.isEraser, true);

lastPosition = newPosition;

}, { passive: false });

ui.drawingCanvas.addEventListener('touchend', () => {

if (!isDrawer) return;

isDrawing = false;

// Push any remaining cache items

if (drawingCache.length > 0) {

DRAWING\_DOC.update({ lines: firebase.firestore.FieldValue.arrayUnion(...drawingCache) })

.then(() => { drawingCache = []; })

.catch(err => console.error("Error writing drawing data: ", err));

}

});

// Tool Selection

document.querySelectorAll('.tool-btn').forEach(button => {

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

const tool = button.getAttribute('data-tool');

if (tool === 'eraser') {

currentTool.isEraser = true;

currentTool.width = 15; // Thicker for eraser

} else if (tool === 'brush') {

currentTool.isEraser = false;

currentTool.color = button.getAttribute('data-color');

currentTool.width = 5;

}

// No full Undo/Redo/Fill implementation for simplicity

});

});

// Set default tool

currentTool.color = document.querySelector('.tool-btn[data-color="#000000"]').getAttribute('data-color');