import { initializeApp } from "firebase/app";

import { getAuth, signInAnonymously, onAuthStateChanged } from "firebase/auth";

import { getFirestore, collection, doc, setDoc, updateDoc, onSnapshot, query, orderBy, where, serverTimestamp, FieldValue } from "firebase/firestore";

import { getDatabase, ref, set as setRTDB, onValue as onValueRTDB } from "firebase/database";

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

// Replace with your actual Firebase Configuration

const firebaseConfig = {

apiKey: "AIzaSyBhyOdPB98l0SBgUduxunjx5s3HJBgWyQM",

authDomain: "classroompictionaryapp.firebaseapp.com",

projectId: "classroompictionaryapp",

storageBucket: "classroompictionaryapp.firebasestorage.app",

messagingSenderId: "812924722367",

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

databaseURL: "https://classroompictionaryapp-default-rtdb.firebaseio.com", // \*\*NEW: RTDB URL\*\*

};

// Initialize Firebase services

const app = initializeApp(firebaseConfig);

// Firestore (for structured data: game state, users, attempts)

const db = getFirestore(app);

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

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

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

// Realtime Database (RTDB) (for high-speed drawing data)

const dbRT = getDatabase(app);

const DRAWING\_RTDB\_REF = ref(dbRT, 'drawing/currentDrawing');

// Authentication

const auth = getAuth(app);

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

let currentUserID = null; // \*\*SECURE ID\*\* - Set after successful authentication

let username = null;

let gameStatus = 'LOBBY';

let isDrawer = false;

let isTeacher = false; // \*\*SECURE ROLE STATE\*\*

let lastDrawingData = null; // Cache of the last received drawing state

// Drawing State

let isDrawing = false;

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

let drawingCache = []; // Batching drawing segments before sending to RTDB

// Drawing History for Undo/Redo

const canvasHistory = [];

let historyPointer = -1;

const MAX\_HISTORY = 30;

// Current Tool Configuration

let currentTool = {

color: '#000000',

width: 5,

isEraser: false

};

const ui = {

// Screens

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

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

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

// Lobby Elements

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'),

// Game Elements

drawerName: document.getElementById('drawer-name'),

timerDisplay: document.getElementById('timer-display'),

wordToDraw: document.getElementById('word-to-draw'),

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

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

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

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

// Canvas Context (defined after initialization)

ctx: null,

// Overlays

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

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

};

// --- SECTION 3: CANVAS & UTILITY FUNCTIONS ---

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

};

}

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

// 1. Draw on the local canvas

if (isEraser) {

ui.ctx.globalCompositeOperation = 'destination-out'; // Erase mode

ui.ctx.lineWidth = width \* 2; // Make eraser visually thicker

} else {

ui.ctx.globalCompositeOperation = 'source-over'; // Normal draw mode

ui.ctx.strokeStyle = color;

ui.ctx.lineWidth = width;

}

ui.ctx.lineCap = 'round';

ui.ctx.beginPath();

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

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

ui.ctx.stroke();

// 2. Queue for database broadcast

if (broadcast && isDrawer) {

drawingCache.push({

start: start,

end: end,

color: color,

width: width,

isEraser: isEraser

});

}

}

function clearCanvas() {

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

}

function showOverlay(message, duration = 3000) {

ui.overlayText.textContent = message;

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

ui.messageOverlay.style.opacity = 1;

setTimeout(() => {

ui.messageOverlay.style.opacity = 0;

setTimeout(() => ui.messageOverlay.classList.add('hidden'), 500);

}, duration);

}

// \*\*NEW:\*\* CANVAS HISTORY FUNCTIONS for UNDO/REDO

function saveCanvasState() {

// Only save history if the user is currently the drawer

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

// If pointer is not at the end, clear "redo" future states

if (historyPointer < canvasHistory.length - 1) {

canvasHistory.length = historyPointer + 1;

}

// Save the current canvas content as a Data URL

const dataURL = ui.drawingCanvas.toDataURL();

canvasHistory.push(dataURL);

historyPointer = canvasHistory.length - 1;

// Trim history if it exceeds the max limit

if (canvasHistory.length > MAX\_HISTORY) {

canvasHistory.shift(); // Remove the oldest state

historyPointer--;

}

}

function loadCanvasState(dataURL) {

const image = new Image();

image.onload = function() {

// Clear and draw the historical state

clearCanvas();

ui.ctx.drawImage(image, 0, 0);

};

image.src = dataURL;

}

function undoCanvas() {

if (!isDrawer || historyPointer <= 0) return;

historyPointer--; // Move one step back

if (canvasHistory[historyPointer]) {

loadCanvasState(canvasHistory[historyPointer]);

// For the RTDB/Network: clear the document to reflect the undo on other clients

setRTDB(DRAWING\_RTDB\_REF, { cleared: true, timestamp: Date.now() });

} else {

// If we undo past the last saved state, clear the canvas entirely

clearCanvas();

// For the RTDB/Network: clear the document to reflect the undo on other clients

setRTDB(DRAWING\_RTDB\_REF, { cleared: true, timestamp: Date.now() });

}

}

// --- SECTION 4: FIREBASE LISTENERS (RTDB & FIRESTORE) ---

function setupFirebaseListeners() {

// Listener 1: Game Session State (Firestore)

onSnapshot(GAME\_SESSION\_DOC, (docSnapshot) => {

if (!docSnapshot.exists()) return;

const session = docSnapshot.data();

gameStatus = session.status;

const currentDrawer = session.drawerId;

const currentWord = session.currentWord;

const timeRemaining = session.timer;

// Determine if the current user is the drawer

isDrawer = (currentUserID === currentDrawer);

// UI State Management

if (gameStatus === 'LOBBY') {

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

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

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

// \*\*SECURE TEACHER CHECK:\*\* Use the state set in Listener 2

ui.teacherControls.classList.toggle('hidden', !isTeacher);

// Clear the board for a new game

clearCanvas();

setRTDB(DRAWING\_RTDB\_REF, { cleared: true, timestamp: Date.now() });

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

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

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

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

ui.timerDisplay.textContent = `Time: ${timeRemaining}s`;

// Drawing UI

if (isDrawer) {

ui.wordToDraw.textContent = `Your Word: ${currentWord}`;

ui.wordToDraw.style.backgroundColor = 'var(--accent-color)';

ui.drawingCanvas.style.pointerEvents = 'auto'; // Enable drawing

ui.guessInput.disabled = true;

ui.submitGuessBtn.disabled = true;

} else {

ui.wordToDraw.textContent = `Word: ????`;

ui.wordToDraw.style.backgroundColor = 'var(--primary-color)';

ui.drawingCanvas.style.pointerEvents = 'none'; // Disable drawing

ui.guessInput.disabled = false;

ui.submitGuessBtn.disabled = false;

}

}

});

// Listener 2: Users and Live Leaderboard (Firestore)

onSnapshot(query(USERS\_COLLECTION, orderBy('currentPoints', 'desc')), (snapshot) => {

let leaderboardHtml = '';

ui.attemptsList.innerHTML = '';

isTeacher = false; // Reset teacher status

snapshot.forEach(doc => {

const user = doc.data();

// \*\*SECURE ROLE CHECK:\*\* Update local state from Firestore document

if (doc.id === currentUserID) {

isTeacher = (user.role === 'teacher');

}

leaderboardHtml += `

<li>

${user.username}: <span>${user.currentPoints} pts</span>

</li>

`;

});

document.getElementById('live-leaderboard-list').innerHTML = leaderboardHtml;

});

// Listener 3: Attempts/Chat (Firestore)

onSnapshot(query(ATTEMPTS\_COLLECTION, orderBy('timestamp', 'desc')), (snapshot) => {

let attemptsHtml = '';

snapshot.forEach(doc => {

const attempt = doc.data();

const cssClass = attempt.isCorrect ? 'correct-guess' : '';

attemptsHtml += `

<li class="${cssClass}">

\*\*${attempt.username}\*\*: ${attempt.guess}

</li>

`;

});

ui.attemptsList.innerHTML = attemptsHtml;

});

// \*\*NEW:\*\* Listener 4: Realtime Drawing Data (RTDB)

onValueRTDB(DRAWING\_RTDB\_REF, (snapshot) => {

if (!isDrawer) { // Only clients who are NOT drawing listen and render

const data = snapshot.val();

if (data && data.cleared) {

// If a clear or undo signal is sent, clear the canvas locally

clearCanvas();

lastDrawingData = null;

return;

}

if (data && data.lines) {

// To prevent redrawing the entire line array on every tiny update,

// we only draw the difference from the last known state.

const newLines = data.lines.slice(lastDrawingData ? lastDrawingData.lines.length : 0);

newLines.forEach(line => {

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

});

// Cache the current data state

lastDrawingData = data;

}

}

});

}

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

function initializeGame(uid) {

currentUserID = uid;

// Initialize Canvas Context

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

// Start all Firebase listeners

setupFirebaseListeners();

// Force the client to the Lobby screen initially

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

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

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

console.log("Game initialized with secure UID:", currentUserID);

}

// 1. Join Game

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

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

if (inputUsername.length < 3) {

showOverlay("Please enter a username of at least 3 characters.");

return;

}

username = inputUsername;

// \*\*SECURE USER CREATION/UPDATE:\*\* Use secure ID and merge to protect the 'role' field

await setDoc(doc(USERS\_COLLECTION, currentUserID), {

userID: currentUserID,

username: username,

currentPoints: 0,

// role field is intentionally omitted and must be set manually for teacher access

}, { merge: true });

// Hide auth area and show active user name

document.getElementById('auth-area').classList.add('hidden');

document.getElementById('active-user-display').textContent = `User: ${username}`;

});

// 2. Teacher Controls

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

if (!isTeacher) { // Double-check security

showOverlay("You must be the teacher to start the game.");

return;

}

const prompts = ui.promptListInput.value.trim().split('\n').