# Project Brief: Technical Support Self-Service Kiosk

Version: 1.0

Date: August 16, 2025

Primary Stakeholders: Christy Fisher (CTO), Kevin Hogan (Assistant Director)

## 1. Project Overview

### 1.1. Mission Statement

To streamline the IT support process at Norman Public Schools by providing an automated, user-friendly kiosk for students and staff to report technical issues. This kiosk will reduce the burden on support technicians for initial intake, improve the quality of information gathered, and accelerate ticket resolution times.

### 1.2. Goals

* **Automate Intake:** Reduce the time technicians spend on initial user identification and problem description.
* **Improve Data Quality:** Use a conversational AI to ask clarifying questions and generate concise, actionable problem summaries.
* **Increase Efficiency:** Automatically create detailed tickets in Incident IQ, complete with user information, asset details, and video evidence.
* **Enhance User Experience:** Provide a simple, accessible, and modern interface for users to report issues without waiting for a technician.

### 1.3. Target Environment

The application is designed to run on dedicated kiosk hardware at 6 school locations. It must be robust enough to handle a high volume of daily use (estimated max of 30 uses/day per school) and function effectively in potentially noisy environments like hallways or libraries.

## 2. Core Features & Functions

The application operates as a state machine, guiding the user through a logical sequence of steps to report an issue.

### 2.1. User Flow

1. **Start:** The kiosk displays a welcome screen. The user taps the screen to begin.
2. **Identification:** The camera and microphone activate. The user is prompted to scan their ID badge. A visual scanner box provides a clear target.
3. **Voice Fallback:** If scanning fails or is not possible, the user can press and hold a "Hold to Speak" button to state their name or ID number.
4. **User Verification:**
   * If a single user match is found, the kiosk displays their name and asks for confirmation via "Yes/No" buttons.
   * If multiple matches are found, a list of potential users is displayed for selection.
5. **Asset Selection:** The kiosk fetches and displays a list of assets assigned to the verified user in Incident IQ. The user selects the problematic device or chooses "It's something else."
6. **Problem Description:** The user is prompted to "Hold to Speak" and describe their issue.
7. **AI Clarification:** The conversational AI analyzes the initial description.
   * If the description is vague, the AI will ask up to **two** clarifying questions, remembering the full context (user, device, previous answers) at each step.
   * Once the problem is clear or the question limit is reached, the AI generates a final, concise summary.
8. **Ticket Confirmation:** The user is shown a summary of the ticket details (Name, Device, and AI-generated Problem Description). They can confirm by clicking "Yes, Create Ticket" or restart the problem description process by clicking "No, Let's Try Again."
9. **Ticket Creation:** Upon confirmation, the app creates a ticket in Incident IQ, including all relevant IDs, tags, and a link to a video recording.
10. **Confirmation Screen:** A final screen displays the new ticket number and a thank you message. The session automatically resets after 10 seconds.

### 2.2. Key Features

* **Barcode & Voice Identification:** Dual methods for user lookup.
* **Dynamic Asset Listing:** Pulls real-time asset data from Incident IQ.
* **Conversational AI Intake:** A smart, context-aware assistant that asks relevant follow-up questions and enforces a two-question limit.
* **Push-to-Talk Interface:** Ensures high-accuracy voice input by only listening when the user is actively pressing the button.
* **Noise Suppression:** Utilizes browser-based audio processing to filter background noise.
* **Direct Incident IQ Integration:** Creates fully categorized tickets with all necessary information.
* **Video Submission:** Records the user's description and attaches a (simulated) link.
* **Debug Mode:** An icon allows developers to view the exact prompts being sent to the AI for troubleshooting.

## 3. Technical Documentation

### 3.1. Architecture & Stack

* **Frontend Framework:** React.js
* **Styling:** Tailwind CSS
* **Backend Services:**
  + **Firebase:** Used for anonymous user authentication for the kiosk session and for logging ticket data to a Firestore database for analytics.
  + **Google Generative AI (Gemini):** The gemini-2.0-flash model is used for all natural language processing tasks.
  + **Incident IQ API:** The primary integration point for user data, asset data, and ticket creation.
* **Browser APIs:**
  + getUserMedia: Accesses the camera and microphone. noiseSuppression and echoCancellation are enabled.
  + MediaRecorder: Records video from the user's camera stream.
  + BarcodeDetector: Scans for barcodes in the video stream.
  + SpeechRecognition: Handles voice-to-text transcription.

### 3.2. Key API Endpoints (Incident IQ)

* **User Search:** GET /api/v1.0/users/{searchTerm} and GET /services/users?$filter=(SearchText contains '{name}')
* **Asset Search:** GET /api/v1.0/assets?$filter=...&$expand=Model
* **Category & Issue Fetching:** GET /api/v1.0/categories/v2 and GET /api/v1.0/public/issue-catalog/issues?categoryId={id}
* **Ticket Creation:** POST /api/v1.0/tickets/new

### 3.3. Core Logic & Functions

* **App Component:** The main component that manages the overall application state and renders all other components.
* **LiveStatusDisplay Component:** A versatile component that renders the appropriate UI for each step of the user flow.
* **getProblemSolvingResponse():** The central AI function that maintains conversational context, decides whether to ask a follow-up question or summarize, and enforces the two-question limit.
* **prepareTicket():** Gathers all the necessary information (user ID, asset ID, category IDs, AI summary) and formats it into the JSON payload required by the Incident IQ API. It correctly maps the AI-selected issue to the IssueId, IssueCategoryId, and IssueTypeId fields.
* **createTicket():** Sends the final, formatted payload to the Incident IQ API to create the ticket.

## 4. Supporting Code

Below is the complete, most up-to-date code for the application. This code is self-contained and can be used to relaunch the project in a co-authoring session.

import React, { useState, useEffect, useRef } from 'react';  
import { initializeApp } from 'firebase/app';  
import { getFirestore, collection, addDoc, setLogLevel } from 'firebase/firestore';  
import { getAuth, signInAnonymously, onAuthStateChanged, signInWithCustomToken } from 'firebase/auth';  
  
// --- Helper Icons & Logo (as SVG/React components) ---  
const MicIcon = ({ className }) => (  
 <svg className={className} xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="currentColor">  
 <path d="M12 12c2.21 0 4-1.79 4-4V4c0-2.21-1.79-4-4-4S8 1.79 8 4v4c0 2.21 1.79 4 4 4zm-2-4c0-1.1.9-2 2-2s2 .9 2 2v4c0 1.1-.9 2-2 2s-2-.9-2-2V8zm10 4h-2c0 3.31-2.69 6-6 6s-6-2.69-6-6H4c0 4.42 3.58 8 8 8v3h2v-3c4.42 0 8-3.58 8-8z"/>  
 </svg>  
);  
  
const CheckCircleIcon = ({ className }) => (  
 <svg className={className} xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="currentColor">  
 <path fillRule="evenodd" d="M2.25 12c0-5.385 4.365-9.75 9.75-9.75s9.75 4.365 9.75 9.75-4.365 9.75-9.75 9.75S2.25 17.385 2.25 12zm13.36-1.814a.75.75 0 10-1.22-.872l-3.236 4.53L9.53 12.22a.75.75 0 00-1.06 1.06l2.25 2.25a.75.75 0 001.14-.094l3.75-5.25z" clipRule="evenodd" />  
 </svg>  
);  
  
const SparkleIcon = ({ className }) => (  
 <svg className={className} xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="currentColor">  
 <path d="M12 2.25a.75.75 0 01.75.75v3a.75.75 0 01-1.5 0v-3a.75.75 0 01.75-.75zM16.03 5.97a.75.75 0 011.06 0l2.122 2.121a.75.75 0 01-1.06 1.06L16.03 7.03a.75.75 0 010-1.06zM21 11.25a.75.75 0 010 1.5h-3a.75.75 0 010-1.5h3zM16.03 16.03a.75.75 0 010 1.06l-2.121 2.122a.75.75 0 01-1.06-1.06l2.121-2.122a.75.75 0 011.06 0zM12 18.75a.75.75 0 01-.75.75v3a.75.75 0 011.5 0v-3a.75.75 0 01-.75-.75zM7.97 16.03a.75.75 0 01-1.06 0l-2.122-2.121a.75.75 0 011.06-1.06L7.97 14.97a.75.75 0 010 1.06zM3 12.75a.75.75 0 010-1.5h3a.75.75 0 010 1.5H3zM7.97 7.97a.75.75 0 010-1.06l2.121-2.122a.75.75 0 011.06 1.06L9.03 7.97a.75.75 0 01-1.06 0z" />  
 </svg>  
);  
  
const LoadingSpinner = () => (  
 <div className="flex justify-center items-center gap-2">  
 <div className="w-3 h-3 rounded-full bg-cyan-400 animate-pulse [animation-delay:-0.3s]"></div>  
 <div className="w-3 h-3 rounded-full bg-cyan-400 animate-pulse [animation-delay:-0.15s]"></div>  
 <div className="w-3 h-3 rounded-full bg-cyan-400 animate-pulse"></div>  
 </div>  
);  
  
const NTechLogo = () => (  
 <img src="data:image/png;base64," alt="N-Tech Logo" className="h-16 w-auto" />  
);  
  
const DebugIcon = ({ className }) => (  
 <svg className={className} xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" fill="currentColor">  
 <path fillRule="evenodd" d="M12.96 6.22a.75.75 0 0 1 1.06 0l.15.15a3.75 3.75 0 0 0 5.3 0s.15-.15.15-.15a.75.75 0 1 1 1.06 1.06l-.15.15a5.25 5.25 0 0 1-7.42 0l-.15-.15a.75.75 0 0 1 0-1.06ZM11.04 6.22a.75.75 0 0 0-1.06 0l-.15.15a3.75 3.75 0 0 1-5.3 0s-.15-.15-.15-.15a.75.75 0 1 0-1.06 1.06l.15.15a5.25 5.25 0 0 0 7.42 0l.15-.15a.75.75 0 0 0 0-1.06ZM4.5 12.75a.75.75 0 0 0 0 1.5h15a.75.75 0 0 0 0-1.5H4.5Z" clipRule="evenodd" />  
 <path d="M3.75 9.75a.75.75 0 0 0 0 1.5h.563c.24 1.583 1.13 2.828 2.388 3.494a.75.75 0 1 0 .698-1.298 2.5 2.5 0 0 1-1.63-2.196h1.03c.24 1.583 1.13 2.828 2.388 3.494a.75.75 0 1 0 .698-1.298 2.5 2.5 0 0 1-1.63-2.196h1.03c.24 1.583 1.13 2.828 2.388 3.494a.75.75 0 1 0 .698-1.298 2.5 2.5 0 0 1-1.63-2.196h1.03c.24 1.583 1.13 2.828 2.388 3.494a.75.75 0 1 0 .698-1.298 2.5 2.5 0 0 1-1.63-2.196H19.5a.75.75 0 0 0 0-1.5H3.75Z" />  
 </svg>  
);  
  
const ScannerBox = () => (  
 <div className="relative w-64 h-40 sm:w-80 sm:h-36 my-4">  
 <div className="w-full h-full border-2 border-dashed border-cyan-400/50 rounded-lg"></div>  
 <div className="absolute top-0 left-0 w-full h-1 bg-cyan-400 shadow-[0\_0\_10px\_theme(colors.cyan.400)] animate-scan"></div>  
 <div className="absolute top-0 left-0 w-8 h-8 border-t-4 border-l-4 border-cyan-400 rounded-tl-lg"></div>  
 <div className="absolute top-0 right-0 w-8 h-8 border-t-4 border-r-4 border-cyan-400 rounded-tr-lg"></div>  
 <div className="absolute bottom-0 left-0 w-8 h-8 border-b-4 border-l-4 border-cyan-400 rounded-bl-lg"></div>  
 <div className="absolute bottom-0 right-0 w-8 h-8 border-b-4 border-r-4 border-cyan-400 rounded-br-lg"></div>  
 </div>  
);  
  
  
// --- Main App Component ---  
export default function App() {  
 // --- API Keys and Configuration ---  
 const GEMINI\_API\_KEY = 'AIzaSyDMZ-WWv0ejKMm2P1dmtVPs2vsGUxkYxlA';  
 const INCIDENT\_IQ\_API\_TOKEN = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiIxZTIzMTcwYS0yZTFiLTQ5Y2QtYjZhNi0yZDlmOWUxMmE4OTIiLCJzY29wZSI6Imh0dHBzOi8vbm9ybWFucHMuaW5jaWRlbnRpcS5jb20iLCJzdWIiOiJkNTEzMmNmNS0wYzg1LTQ1YjctOGE5Ni0wYzE4NDkyZmIwY2IiLCJqdGkiOiIzYzI4YmE2Zi0yZTVjLWYwMTEtOGY3Yy0wMDBkM2EwZGI5YTciLCJpYXQiOjE3NTIwMDE1MDUuNjM3LCJleHAiOjE4NDY2OTU5MDUuNjQzfQ.Y7FllRXpjLBwuRZ2KGmhl8BeLa3ewL-j9cdJjGCOFD4';  
 const INCIDENT\_IQ\_DISTRICT\_URL = 'https://normanps.incidentiq.com';  
 const SITE\_ID = '1e23170a-2e1b-49cd-b6a6-2d9f9e12a892';  
 const TICKETING\_PRODUCT\_ID = ''; // TODO: Replace with actual Product ID from Incident IQ  
   
 // --- State Management ---  
 const [status, setStatus] = useState('awaiting\_start');  
 const [visitorName, setVisitorName] = useState('');  
 const [problemDescription, setProblemDescription] = useState('');  
 const [troubleshootingSuggestion, setTroubleshootingSuggestion] = useState('');  
 const [qaResponse, setQaResponse] = useState('');  
 const [identifiedAsset, setIdentifiedAsset] = useState(null);  
 const [iiqUser, setIiqUser] = useState(null);  
 const [ticketDetails, setTicketDetails] = useState(null);  
 const [isListening, setIsListening] = useState(false);  
 const [interimTranscript, setInterimTranscript] = useState('');  
 const [finalTicket, setFinalTicket] = useState(null);  
 const [errorMessage, setErrorMessage] = useState('');  
 const [potentialUser, setPotentialUser] = useState(null);  
 const [potentialUsers, setPotentialUsers] = useState([]);  
 const [userAssets, setUserAssets] = useState([]);  
 const [conversationHistory, setConversationHistory] = useState([]);  
 const [clarificationCount, setClarificationCount] = useState(0);  
 const [clarificationQuestion, setClarificationQuestion] = useState('');  
 const [showDebug, setShowDebug] = useState(false);  
 const [debugPrompt, setDebugPrompt] = useState('');  
 const [iiqCategories, setIiqCategories] = useState([]);  
   
 // --- Firebase State ---  
 const [db, setDb] = useState(null);  
 const [auth, setAuth] = useState(null);  
 const [appId, setAppId] = useState('default-kiosk-app');  
  
 // --- Refs ---  
 const videoRef = useRef(null);  
 const recognitionRef = useRef(null);  
 const mediaRecorderRef = useRef(null);  
 const recordedChunksRef = useRef([]);  
 const barcodeDetectorRef = useRef(null);  
 const processTranscriptTimeoutRef = useRef(null);  
 const resetSessionTimeoutRef = useRef(null);  
 const scanIntervalRef = useRef(null);  
 const statusRef = useRef(status);  
 const isRecognitionActiveRef = useRef(false);  
 const permissionDeniedRef = useRef(false);  
 const finalTranscriptRef = useRef('');  
  
 // Use a ref to hold the latest transcript processing function  
 const processTranscriptRef = useRef();  
 useEffect(() => {  
 processTranscriptRef.current = processTranscript;  
 });  
  
 useEffect(() => {  
 statusRef.current = status;  
 // Stop recognition when we show the final confirmation buttons  
 const buttonDrivenStates = [  
 'awaiting\_id\_confirmation',  
 'awaiting\_barcode\_confirmation',  
 'awaiting\_confirmation'  
 ];  
 if (buttonDrivenStates.includes(status) && recognitionRef.current && isRecognitionActiveRef.current) {  
 console.log(`Stopping recognition for button-driven state: ${status}.`);  
 recognitionRef.current.stop();  
 }  
 }, [status]);  
  
 // --- Firebase Initialization ---  
 useEffect(() => {  
 try {  
 const firebaseConfig = typeof \_\_firebase\_config !== 'undefined' ? JSON.parse(\_\_firebase\_config) : {};  
 const currentAppId = typeof \_\_app\_id !== 'undefined' ? \_\_app\_id : 'default-kiosk-app';  
 if (Object.keys(firebaseConfig).length > 0) {  
 const app = initializeApp(firebaseConfig);  
 const firestoreDb = getFirestore(app);  
 const firebaseAuth = getAuth(app);  
 setDb(firestoreDb); setAuth(firebaseAuth); setAppId(currentAppId);  
 setLogLevel('debug');  
 onAuthStateChanged(firebaseAuth, user => {  
 if (!user) {  
 const token = typeof \_\_initial\_auth\_token !== 'undefined' ? \_\_initial\_auth\_token : null;  
 if (token) {  
 signInWithCustomToken(firebaseAuth, token).catch(e => {   
 console.error("Custom token sign-in failed, falling back to anonymous.", e);   
 signInAnonymously(firebaseAuth);   
 });  
 } else {   
 signInAnonymously(firebaseAuth);   
 }  
 }  
 });  
 } else { console.warn("Firebase config not found."); }  
 } catch (e) {  
 console.error("Firebase init error:", e);  
 setStatus('error'); setErrorMessage('System configuration error.');  
 }  
 }, []);  
  
 // --- Fetch Categories on Load ---  
 useEffect(() => {  
 const fetchCategories = async () => {  
 const categories = await getIiqCategories();  
 setIiqCategories(categories);  
 };  
  
 fetchCategories();  
 }, []);  
  
 // --- Main Session Reset Logic ---  
 const resetSession = () => {  
 setVisitorName('');  
 setProblemDescription('');  
 setTroubleshootingSuggestion('');  
 setIdentifiedAsset(null);  
 setIiqUser(null);  
 setTicketDetails(null);  
 setQaResponse('');  
 setInterimTranscript('');  
 setFinalTicket(null);  
 setErrorMessage('');  
 setPotentialUser(null);  
 setPotentialUsers([]);  
 setUserAssets([]);  
 setConversationHistory([]);  
 setClarificationCount(0);  
 setClarificationQuestion('');  
 permissionDeniedRef.current = false;  
 setStatus('awaiting\_start');  
 if (recognitionRef.current) {  
 recognitionRef.current.stop();  
 }  
 if (scanIntervalRef.current) clearInterval(scanIntervalRef.current);  
 };  
  
 // --- Handle "Try Again" from multi-user selection ---  
 const handleTryAgain = () => {  
 setPotentialUser(null);  
 setPotentialUsers([]);  
 setStatus('awaiting\_name');  
 setupSpeechRecognition();  
 };  
   
 // --- Handle restarting the problem description ---  
 const handleRedoProblem = () => {  
 setProblemDescription('');  
 setTroubleshootingSuggestion('');  
 setTicketDetails(null);  
 setConversationHistory([]);  
 setClarificationCount(0);  
 setClarificationQuestion('');  
 setStatus('awaiting\_problem');  
 setupSpeechRecognition();  
 };  
  
 // --- Start Camera and Barcode Scanner ---  
 const startBarcodeScanner = () => {  
 if (!barcodeDetectorRef.current) {  
 console.log("Barcode detector not ready, cannot start scanner.");  
 return;  
 }  
 if (scanIntervalRef.current) clearInterval(scanIntervalRef.current);  
 scanIntervalRef.current = setInterval(async () => {  
 if (videoRef.current?.readyState >= 4 && (statusRef.current === 'awaiting\_scan' || statusRef.current === 'awaiting\_name')) {  
 try {  
 const barcodes = await barcodeDetectorRef.current.detect(videoRef.current);  
 if (barcodes.length > 0) {  
 const scannedId = barcodes[0].rawValue;  
 if(scanIntervalRef.current) clearInterval(scanIntervalRef.current);  
 verifyUserByBarcode(scannedId);  
 }  
 } catch (e) { console.error("Error during barcode detection:", e); if(scanIntervalRef.current) clearInterval(scanIntervalRef.current); }  
 }  
 }, 1000);  
 }  
  
 // --- Setup and Start Speech Recognition ---  
 const setupSpeechRecognition = () => {  
 if (recognitionRef.current) {  
 return;  
 }  
 const SpeechRecognition = window.SpeechRecognition || window.webkitSpeechRecognition;  
 if (!SpeechRecognition) {  
 setStatus('error');  
 setErrorMessage('Speech recognition not supported.');  
 return;  
 }  
 const recognition = new SpeechRecognition();  
 recognition.continuous = true;  
 recognition.interimResults = true;  
 recognition.lang = 'en-US';  
  
 recognition.onstart = () => {  
 isRecognitionActiveRef.current = true;  
 setIsListening(true);  
 if (mediaRecorderRef.current?.state === 'inactive') startRecording();  
 };  
  
 recognition.onend = () => {  
 isRecognitionActiveRef.current = false;  
 setIsListening(false);  
 console.log("Recognition ended for status:", statusRef.current);  
 if (finalTranscriptRef.current) {  
 processTranscriptRef.current(finalTranscriptRef.current.trim());  
 finalTranscriptRef.current = ''; // Clear after processing  
 }  
 };  
  
 recognition.onerror = (event) => {  
 isRecognitionActiveRef.current = false;  
 setIsListening(false);  
 if (event.error === 'not-allowed') {  
 permissionDeniedRef.current = true;  
 setErrorMessage("Microphone access was denied. Please enable it in your browser settings and tap the screen to restart.");  
 setStatus('error');  
 return;  
 }  
 if (event.error === 'no-speech' || event.error === 'aborted') {  
 console.log("No speech detected or recognition aborted.");  
 return;  
 }  
 console.error("Critical speech recognition error:", event.error, event.message);  
 setErrorMessage(`Speech recognition error: ${event.error}.`);  
 };  
  
 recognition.onresult = (event) => {  
 let final = '';  
 let interim = '';  
 for (let i = event.resultIndex; i < event.results.length; ++i) {  
 if (event.results[i].isFinal) {  
 final += event.results[i][0].transcript;  
 } else {  
 interim += event.results[i][0].transcript;  
 }  
 }  
 setInterimTranscript(interim);  
 if (final) {  
 finalTranscriptRef.current += final + ' ';  
 }  
 };  
  
 recognitionRef.current = recognition;  
 };  
   
 // --- AI-Powered Problem Clarification and Summarization ---  
 const getProblemSolvingResponse = async (history, asset, count, userName) => {  
 const assetInfo = asset ? `The user is having a problem with their ${asset.Name} (Model: ${asset.Model?.Name || 'N/A'}).` : "The user has not specified a device.";  
   
 const prompt = `You are an expert IT support technician helping a user named ${userName}. Your goal is to gather information to create a useful support ticket.  
  
\*\*CONTEXT:\*\*  
- User: ${userName}  
- Device: ${assetInfo}  
- Conversation History:  
 ${history.map(h => `${h.role === 'user' ? userName : 'Assistant'}: ${h.parts[0].text}`).join('\n')}  
- Questions Asked So Far: ${count}  
  
\*\*YOUR TASK (Follow these steps in order):\*\*  
1. \*\*Analyze Completeness:\*\* Review the entire conversation history. Do you have a specific, actionable problem description? "It's broken" is not enough. "The screen is cracked" is enough.  
2. \*\*Decision:\*\*  
 - \*\*IF\*\* the information is complete \*\*OR\*\* if you have already asked 2 questions (the "Questions Asked So Far" is 2), you MUST proceed to Step 4 (Summarize).  
 - \*\*ELSE\*\* (the information is vague and you have asked fewer than 2 questions), proceed to Step 3 (Ask).  
3. \*\*Ask:\*\* Formulate ONE clarifying question. Do not repeat previous questions. The goal is to get a more specific detail.  
4. \*\*Summarize:\*\* Write a concise, one-paragraph summary of the issue based on ALL information gathered.  
  
\*\*RESPONSE FORMAT:\*\*  
You MUST respond with a valid JSON object.  
- If you decided to ask a question in Step 3, use this format:  
 \`{"status": "needs\_clarification", "content": "Your question here."}\`  
- If you decided to summarize in Step 4, use this format:  
 \`{"status": "complete", "content": "Your summary paragraph here."}\``;  
   
 setDebugPrompt(prompt); // Update debug state  
 const payload = {  
 contents: [{ role: "user", parts: [{ text: prompt }] }],  
 generationConfig: {  
 responseMimeType: "application/json",  
 }  
 };  
 try {  
 const apiKey = GEMINI\_API\_KEY;  
 const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=${apiKey}`;  
 const response = await fetch(apiUrl, {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify(payload)  
 });  
 if (!response.ok) {  
 throw new Error(`API Error: ${response.status}`);  
 }  
 const result = await response.json();  
 const jsonText = result.candidates[0].content.parts[0].text;  
 return JSON.parse(jsonText);  
 } catch (error) {  
 console.error("Clarification/Summarization AI error:", error);  
 const lastUserMessage = history.filter(h => h.role === 'user').pop()?.parts[0]?.text || "Problem description could not be processed.";  
 return { status: 'complete', content: lastUserMessage };  
 }  
 }  
  
 const handleClarificationResponse = (result, history) => {  
 if (result.status === 'needs\_clarification') {  
 setClarificationQuestion(result.content);  
 setConversationHistory([...history, { role: 'model', parts: [{ text: result.content }] }]);  
 setStatus('awaiting\_clarification');  
 } else { // status is 'complete'  
 setProblemDescription(result.content); // This will trigger the useEffect to call prepareTicket  
 }  
 };  
  
 const startClarificationProcess = async (initialProblem) => {  
 setStatus('processing');  
 const initialHistory = [{ role: 'user', parts: [{ text: initialProblem }] }];  
 setConversationHistory(initialHistory);  
 setClarificationCount(0); // Reset count for new problem  
   
 const result = await getProblemSolvingResponse(initialHistory, identifiedAsset, 0, visitorName);  
 handleClarificationResponse(result, initialHistory);  
 };  
  
 // --- Process Transcript with Gemini ---  
 const processTranscript = async (transcript) => {  
 if (!transcript) return;  
  
 const cancelWords = ['cancel', 'start over', 'delete', 'never mind','end'];  
 if (cancelWords.some(word => transcript.toLowerCase().includes(word))) {  
 resetSession();  
 return;  
 }  
   
 const currentStatus = statusRef.current;  
  
 if (currentStatus === 'awaiting\_scan' || currentStatus === 'awaiting\_name') {  
 verifyUserByName(transcript);  
 return;  
 }  
   
 if (currentStatus === 'awaiting\_problem') {  
 startClarificationProcess(transcript);  
 return;  
 }  
   
 if (currentStatus === 'awaiting\_clarification') {  
 const newCount = clarificationCount + 1;  
 setClarificationCount(newCount);  
 const updatedHistory = [...conversationHistory, { role: 'user', parts: [{ text: transcript }] }];  
 setConversationHistory(updatedHistory);  
   
 const result = await getProblemSolvingResponse(updatedHistory, identifiedAsset, newCount, visitorName);  
 handleClarificationResponse(result, updatedHistory);  
 return;  
 }  
 };  
   
 // --- Verify User by Barcode ---  
 const verifyUserByBarcode = async (searchTerm) => {  
 setStatus('verifying');  
 const users = await findUserInIncidentIQ(searchTerm);  
 if (users && users.length > 0) {  
 setErrorMessage(''); // Clear error on success  
 if (users.length === 1) {  
 setPotentialUser(users[0]);  
 setStatus('awaiting\_barcode\_confirmation');  
 } else {  
 setPotentialUsers(users);  
 setStatus('awaiting\_selection');  
 }  
 } else {  
 setPotentialUser(null);  
 setErrorMessage("I couldn't verify that ID. Please try again or tap the screen to enter your name.");  
 setStatus('awaiting\_name');   
 setupSpeechRecognition();  
 }  
 };  
  
 // --- Verify User by Spoken Name ---  
 const verifyUserByName = async (name) => {  
 setStatus('verifying');  
 const users = await findUserInIncidentIQ(name);  
 if (users && users.length > 0) {  
 setErrorMessage(''); // Clear error on success  
 if (users.length > 1) {  
 setPotentialUsers(users);  
 setStatus('awaiting\_selection');  
 } else {  
 setPotentialUser(users[0]);  
 setStatus('awaiting\_id\_confirmation');  
 }  
 } else {  
 setPotentialUser(null);  
 setErrorMessage(`I couldn't find anyone named "${name}". Please try spelling it out.`);  
 setStatus('awaiting\_name');  
 }  
 };  
  
 // --- Handle User Selection from a list ---  
 const handleUserSelection = async (user) => {  
 if (!user) {  
 console.error("handleUserSelection called with null user.");  
 resetSession();  
 return;  
 }  
 const assets = await getUserAssets(user.UserId);  
 const userWithAssets = { ...user, Assets: assets || [] };  
  
 const firstName = toProperCase(userWithAssets.Name.split(' ')[0]);  
 setIiqUser(userWithAssets);  
 setVisitorName(firstName);  
 setPotentialUsers([]);  
 setPotentialUser(null);  
 if (assets && assets.length > 0) {  
 setUserAssets(assets);  
 setStatus('awaiting\_asset\_selection');  
 } else {  
 setStatus('awaiting\_problem');  
 setupSpeechRecognition();  
 }  
 };  
  
 const handleConfirmation = async (isConfirmed) => {  
 if (isConfirmed) {  
 await handleUserSelection(potentialUser);  
 } else {  
 setPotentialUser(null);  
 setErrorMessage("My mistake. Let's try again.");  
 setStatus('awaiting\_name');  
 setupSpeechRecognition();  
 }  
 };  
   
 // --- Handle Asset Selection ---  
 const handleAssetSelection = (asset) => {  
 setIdentifiedAsset(asset);  
 setStatus('awaiting\_problem');  
 setupSpeechRecognition();  
 };  
  
 // --- Helper function for proper casing ---  
 const toProperCase = (str) => {  
 if (!str) return '';  
 return str.charAt(0).toUpperCase() + str.slice(1).toLowerCase();  
 };  
  
 // --- Prepare ticket details for confirmation ---  
 const prepareTicket = async (user, selectedAsset, finalProblemDescription) => {  
 setStatus('processing');  
 if (!user) {  
 setErrorMessage("User information was lost. Please restart the process.");  
 setStatus('error');  
 return;  
 }  
 try {  
 // This AI call is now just for categorization, not summarization  
 const problemDetails = await getProblemDetailsFromAI(finalProblemDescription, selectedAsset ? [selectedAsset] : []);  
 const deviceName = selectedAsset ? ` - ${selectedAsset.Name}` : '';  
   
 const details = {  
 Subject: `Walk Up - ${user.Location?.Name || 'Unknown Location'}${deviceName}`,  
 IssueDescription: finalProblemDescription, // This is the AI-generated summary  
 ForId: user.UserId,  
 LocationId: user.LocationId,  
 IssueId: problemDetails.issueId,  
 IssueCategoryId: problemDetails.categoryId,  
 IssueTypeId: problemDetails.issueId, // Correctly use the specific issue ID  
 IsUrgent: false,  
 Assets: selectedAsset ? [{ AssetId: selectedAsset.AssetId }] : [],  
 Tags: [{ Name: "Walk Up" }],  
 ProductId: TICKETING\_PRODUCT\_ID,  
 };  
 setTicketDetails(details);  
 setStatus('awaiting\_confirmation');  
 } catch (e) {  
 console.error("Failed to prepare ticket details with AI:", e);  
 setErrorMessage("I had trouble analyzing the problem. Let's create a basic ticket.");  
 const deviceName = selectedAsset ? ` - ${selectedAsset.Name}` : '';  
 const fallbackDetails = {  
 Subject: `Walk Up - ${user.Location?.Name || 'Unknown Location'}${deviceName}`,  
 IssueDescription: finalProblemDescription,  
 ForId: user.UserId,  
 LocationId: user.LocationId,  
 ProductId: TICKETING\_PRODUCT\_ID,  
 Assets: selectedAsset ? [{ AssetId: selectedAsset.AssetId }] : [],  
 Tags: [{ Name: "Walk Up" }],  
 };  
 setTicketDetails(fallbackDetails);  
 setStatus('awaiting\_confirmation');  
 }  
 };  
   
 useEffect(() => {  
 // This effect triggers the ticket preparation process once the AI has generated the final problem summary.  
 if (problemDescription && iiqUser && !ticketDetails) {  
 prepareTicket(iiqUser, identifiedAsset, problemDescription);  
 }  
 }, [iiqUser, problemDescription, identifiedAsset, ticketDetails]);  
   
 // --- Create Ticket after Confirmation ---  
 const createTicket = async () => {  
 if (!ticketDetails || !iiqUser) return;  
 setStatus('processing');  
   
 try {  
 const videoLink = await stopRecording();  
   
 const finalTicketDetails = {  
 ...ticketDetails,  
 // The description from prepareTicket is already the full summary. Just append the video link.  
 IssueDescription: `${ticketDetails.IssueDescription}\n\nVideo Submission: ${videoLink || 'Not available.'}`  
 };  
  
 const newTicket = await createIncidentIQTicket(finalTicketDetails);  
   
 if (newTicket.success) {  
 await logTicketToFirestore({ ...finalTicketDetails, ticketNumber: newTicket.ticketNumber });  
 setFinalTicket(newTicket);  
 setStatus('confirming');  
 if (resetSessionTimeoutRef.current) clearTimeout(resetSessionTimeoutRef.current);  
 resetSessionTimeoutRef.current = setTimeout(resetSession, 10000);  
 } else {  
 setErrorMessage(newTicket.errorMessage || "Failed to create ticket in Incident IQ.");  
 setStatus('error');  
 setTimeout(resetSession, 10000);  
 }  
 } catch (error) {  
 console.error("A critical error occurred while creating the ticket.", error)  
 setErrorMessage("A critical error occurred while creating the ticket.");  
 setStatus('error');  
 setTimeout(resetSession, 10000);  
 }  
 };  
   
 // --- Main screen tap handler ---  
 const handleScreenTap = async () => {  
 if (status === 'awaiting\_start') {  
 setStatus('initializing');  
 try {  
 const stream = await navigator.mediaDevices.getUserMedia({   
 video: true,   
 audio: {  
 noiseSuppression: true,  
 echoCancellation: true  
 }   
 });  
  
 if ('BarcodeDetector' in window) {  
 barcodeDetectorRef.current = new window.BarcodeDetector({ formats: ['code\_128', 'qr\_code', 'ean\_13'] });  
 } else {  
 console.warn("BarcodeDetector API not supported.");  
 }  
  
 if (videoRef.current) {  
 videoRef.current.srcObject = stream;  
 videoRef.current.oncanplay = () => videoRef.current.play();  
 }  
  
 mediaRecorderRef.current = new MediaRecorder(stream, { mimeType: 'video/webm' });  
 mediaRecorderRef.current.ondataavailable = (event) => {  
 if (event.data.size > 0) recordedChunksRef.current.push(event.data);  
 };  
   
 setStatus('awaiting\_scan');  
 startBarcodeScanner();  
 setupSpeechRecognition();  
  
 } catch (err) {  
 console.error("Error accessing media devices:", err.name, err.message);  
 if (err.name === 'NotFoundError') {  
 console.warn("Camera/Mic not found. Degrading to voice-only mode.");  
 setErrorMessage("Camera not found. Switching to voice input.");  
 setStatus('awaiting\_name');  
 setupSpeechRecognition();  
 } else {  
 setStatus('error');  
 let msg = `Could not access Camera/Mic (${err.name}). Please tap to restart.`;  
 if (err.name === 'NotAllowedError' || err.name === 'PermissionDeniedError') {  
 msg = 'Camera/Mic access denied. Please enable permissions in your browser settings and tap to restart.';  
 } else if (err.name === 'NotReadableError') {  
 msg = 'Your camera or microphone is currently in use by another application. Please close it and try again.';  
 }  
 setErrorMessage(msg);  
 }  
 }  
 }  
 };  
   
 // --- Video Recording Functions ---  
 const startRecording = () => {  
 if (mediaRecorderRef.current && mediaRecorderRef.current.state === 'inactive') {  
 recordedChunksRef.current = [];  
 mediaRecorderRef.current.start();  
 }  
 };  
  
 const stopRecording = () => {  
 return new Promise(async (resolve) => {  
 if (mediaRecorderRef.current && mediaRecorderRef.current.state === 'recording') {  
 mediaRecorderRef.current.onstop = async () => {  
 const videoBlob = new Blob(recordedChunksRef.current, { type: 'video/webm' });  
 const fileName = `Kiosk-Recording-${new Date().toISOString().replace(/:/g, '-')}.webm`;  
 const videoLink = await uploadToGoogleDrive(videoBlob, fileName);  
 recordedChunksRef.current = [];  
 resolve(videoLink);  
 };  
 mediaRecorderRef.current.stop();  
 } else {  
 resolve(null); // Resolve with null if not recording  
 }  
 });  
 };  
   
 // --- API & DB Functions ---  
 async function getIiqCategories() {  
 const categoriesUrl = `${INCIDENT\_IQ\_DISTRICT\_URL}/api/v1.0/categories/v2?$s=1000`;  
 const headers = {   
 'Authorization': `Bearer ${INCIDENT\_IQ\_API\_TOKEN}`,  
 'Accept': 'application/json'  
 };  
   
 try {  
 const categoriesResponse = await fetch(categoriesUrl, { method: 'GET', headers });  
 if (!categoriesResponse.ok) {  
 throw new Error(`IIQ Category API error: ${categoriesResponse.status}`);  
 }  
 const categoriesData = await categoriesResponse.json();  
 console.log("Fetched Main Categories:", categoriesData);  
   
 if (categoriesData && categoriesData.Items) {  
 const categoryPromises = categoriesData.Items.map(async (category) => {  
 const issuesUrl = `${INCIDENT\_IQ\_DISTRICT\_URL}/api/v1.0/public/issue-catalog/issues?categoryId=${category.Id}`;  
 const issuesResponse = await fetch(issuesUrl, { method: 'GET', headers });  
   
 if (!issuesResponse.ok) {  
 console.warn(`Could not fetch issues for category ${category.Name}`);  
 return {   
 category: category.Name,  
 categoryId: category.Id,  
 subcategories: []  
 };  
 }  
   
 const issuesData = await issuesResponse.json();  
 console.log(`Fetched Issues for ${category.Name}:`, issuesData);  
 const subcategories = issuesData.Items ? issuesData.Items.map(issue => ({  
 issue: issue.Name,  
 issueId: issue.Id  
 })) : [];  
   
 return {  
 category: category.Name,  
 categoryId: category.Id,  
 subcategories: subcategories  
 };  
 });  
   
 const allCategoriesWithIssues = await Promise.all(categoryPromises);  
 return allCategoriesWithIssues.filter(cat => cat.subcategories.length > 0);  
 }  
 return [];  
 } catch (error) {  
 console.error("Error fetching Incident IQ categories:", error);  
 setErrorMessage("Could not load ticket categories from the server.");  
 return [  
 { category: "General Support", categoryId: "default-cat-id", subcategories: [{ issue: "General Issue", issueId: "default-issue-id" }] }  
 ];  
 }  
 }  
  
 async function getUserAssets(UserId) {  
 if (!UserId) return null;  
 const url = `${INCIDENT\_IQ\_DISTRICT\_URL}/api/v1.0/assets`;  
 const payload = {  
 "Filters": [{  
 "Facet": "User",  
 "Id": UserId  
 }]  
 };  
  
 try {  
 const response = await fetch(url, {  
 method: 'POST',  
 headers: {   
 'Authorization': `Bearer ${INCIDENT\_IQ\_API\_TOKEN}`,  
 'Content-Type': 'application/json',  
 'Accept': 'application/json',  
 'siteid': SITE\_ID,  
 'client': 'ApiClient'  
 },  
 body: JSON.stringify(payload)  
 });  
  
 if (!response.ok) {  
 const errorText = await response.text();  
 console.error("IIQ Asset API Error Response Text:", errorText);  
 throw new Error(`IIQ Asset API error: ${response.status}`);  
 }  
  
 const text = await response.text();  
 if(!text) {  
 return [];  
 }  
  
 const data = JSON.parse(text);  
 return data.Items || [];  
 } catch (error) {  
 console.error("Error fetching user assets from Incident IQ:", error);  
 setErrorMessage("Could not retrieve device list.");  
 return null;  
 }  
 }  
  
 async function findUserInIncidentIQ(searchTerm) {  
 const url = `${INCIDENT\_IQ\_DISTRICT\_URL}/api/v1.0/users/${searchTerm}`;  
 try {  
 const response = await fetch(url, {  
 method: 'GET',  
 headers: {   
 'Authorization': `Bearer ${INCIDENT\_IQ\_API\_TOKEN}`,  
 'Accept': 'application/json'  
 }  
 });  
  
 if (!response.ok) {  
 console.warn(`Direct lookup for ${searchTerm} failed. Falling back to search.`);  
 return findUserByName(searchTerm);  
 }  
  
 const data = await response.json();  
 return [data];   
  
 } catch (error) {  
 console.error("Error with direct user lookup, falling back to search:", error);  
 return findUserByName(searchTerm);  
 }  
 }  
  
 async function findUserByName(name) {  
 const url = `${INCIDENT\_IQ\_DISTRICT\_URL}/services/users?$filter=(SearchText contains '${name}')`;  
 try {  
 const response = await fetch(url, {  
 method: 'GET',  
 headers: {   
 'Authorization': `Bearer ${INCIDENT\_IQ\_API\_TOKEN}`,  
 'Accept': 'application/json'  
 }  
 });  
  
 if (!response.ok) {  
 const errorData = await response.text();  
 console.error("Incident IQ API Error Response:", errorData);  
 throw new Error(`IIQ API error: ${response.status}`);  
 }  
  
 const data = await response.json();  
 if (data && data.Items) {  
 return data.Items;  
 }  
 return [];  
 } catch (error) {  
 console.error("Error fetching user from Incident IQ:", error);  
 return [];  
 }  
 }  
  
 async function getProblemDetailsFromAI(problem, assets) {  
 const assetList = assets && assets.length > 0 ? assets.map(a => ({ AssetId: a.AssetId, Name: a.Name, Model: a.Model?.Name })).join(', ') : 'none';  
 const categoryList = iiqCategories.map(c => ({ category: c.category, categoryId: c.categoryId, subcategories: c.subcategories.map(s => ({ issue: s.issue, issueId: s.issueId })) }));  
  
 const prompt = `Analyze the IT problem: "${problem}".   
 1. From the user's asset list [${assetList}], identify the most relevant asset and return its "AssetId". If no asset is relevant or the list is 'none', return null for assetId.  
 2. From the following category list: ${JSON.stringify(categoryList)}, you MUST choose the best category and subcategory that matches the problem. Return their exact IDs as categoryId and issueId.  
 Respond with only a valid JSON object containing: assetId, categoryId, issueId.`;  
  
   
 const payload = {  
 contents: [{ role: "user", parts: [{ text: prompt }] }],  
 generationConfig: {  
 responseMimeType: "application/json",  
 }  
 };  
   
 const apiKey = GEMINI\_API\_KEY;  
 const apiUrl = `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=${apiKey}`;  
 const response = await fetch(apiUrl, {   
 method: 'POST',   
 headers: { 'Content-Type': 'application/json' },   
 body: JSON.stringify(payload)   
 });  
 if (!response.ok) {  
 const errorBody = await response.text();  
 console.error("AI analysis API failed with status:", response.status, "Body:", errorBody);  
 throw new Error('AI analysis failed');  
 }  
 const result = await response.json();  
 const details = JSON.parse(result.candidates[0].content.parts[0].text);  
   
 if (details.assetId && assets) {  
 const foundAsset = assets.find(a => a.AssetId === details.assetId);  
 if (foundAsset) setIdentifiedAsset(foundAsset);  
 }  
  
 return { assetId: details.assetId, categoryId: details.categoryId, issueId: details.issueId };  
 }  
  
 async function createIncidentIQTicket(ticketData) {  
 console.log("--- Creating REAL Incident IQ Ticket ---", ticketData);  
 const url = `${INCIDENT\_IQ\_DISTRICT\_URL}/api/v1.0/tickets/new`;  
 try {  
 const response = await fetch(url, {  
 method: 'POST',  
 headers: { 'Authorization': `Bearer ${INCIDENT\_IQ\_API\_TOKEN}`, 'Content-Type': 'application/json' }, // <-- 🚨 CRITICAL: This token is exposed on the client-side. In production, this call should be made from a secure backend service.  
 body: JSON.stringify(ticketData)  
 });  
 const responseData = await response.json();  
 if (!response.ok) {  
 console.error("Incident IQ ticket creation failed:", responseData);  
 const errorMessage = responseData.ValidationErrors ? JSON.stringify(responseData.ValidationErrors) : (responseData.Message || "Unknown error creating ticket.");  
 throw new Error(errorMessage);  
 }  
 console.log("Ticket created successfully:", responseData);  
 return {   
 success: true,   
 ticketNumber: responseData.Item.TicketNumber,  
 title: responseData.Item.Subject,  
 visitorName: responseData.Item.For.Name,  
 };  
 } catch (error) {  
 console.error("Error in createIncidentIQTicket:", error);  
 return { success: false, errorMessage: error.message };  
 }  
 }  
  
 async function logTicketToFirestore(ticket) {  
 if (!db) return;  
 try {  
 const docRef = collection(db, `artifacts/${appId}/public/data/tickets`);  
 await addDoc(docRef, { ...ticket, timestamp: new Date().toISOString() });  
 } catch (e) { console.error("Firestore log error: ", e); }  
 }  
   
 async function uploadToGoogleDrive(videoBlob, fileName) {  
 console.log(`--- SIMULATING UPLOAD TO GOOGLE DRIVE ---`);  
 console.log(`File Name: ${fileName}`);  
 console.log(`File Size: ${(videoBlob.size / 1024).toFixed(2)} KB`);  
 console.log(`To implement this, replace this function with a call to a secure backend endpoint.  
That endpoint would use the Google Drive API with a Service Account to upload the file.`);  
 return `https://fake-drive.com/view/${fileName}`; // Return a simulated link  
 }  
  
 return (  
 <div className="w-screen h-screen bg-gray-800 relative flex flex-col overflow-hidden">  
 <style>  
 {`  
 @keyframes scan {  
 0% { top: 0; }  
 100% { top: calc(100% - 4px); }  
 }  
 .animate-scan {  
 animation: scan 2s linear infinite alternate;  
 }  
 `}  
 </style>  
 {status !== 'awaiting\_start' && (  
 <>  
 <video ref={videoRef} playsInline muted className="absolute top-0 left-0 w-full h-full object-cover transform scale-x-[-1]"></video>  
 <div className="absolute inset-0 bg-gradient-to-t from-black/80 via-black/40 to-transparent"></div>  
 <header className="relative z-10 w-full p-8 flex items-center gap-4 bg-black/60 backdrop-blur-sm shadow-lg">  
 <NTechLogo />  
 <div>  
 <h1 className="text-4xl font-bold text-shadow text-white">Tech Support Kiosk</h1>  
 <p className="text-xl text-shadow-sm text-cyan-300">Live Assistant Mode</p>  
 </div>  
 </header>  
 </>  
 )}  
   
 <main className="relative z-10 w-full flex-grow flex flex-col items-center justify-center pb-8 px-8 gap-8">  
 {status === 'awaiting\_start' ? (  
 <div className="text-center cursor-pointer w-full h-full flex flex-col justify-center items-center" onClick={handleScreenTap}>  
 <NTechLogo />  
 <h1 className="text-4xl font-bold text-white mt-4">Tap to Begin</h1>  
 </div>  
 ) : status === 'confirming' && finalTicket ? (  
 <ConfirmationDisplay ticket={finalTicket} />  
 ) : (  
 <LiveStatusDisplay  
 status={status}  
 interimTranscript={interimTranscript}  
 visitorName={visitorName}  
 iiqUser={iiqUser}  
 problemDescription={problemDescription}  
 troubleshootingSuggestion={troubleshootingSuggestion}  
 qaResponse={qaResponse}  
 identifiedAsset={identifiedAsset}  
 isListening={isListening}  
 errorMessage={errorMessage}  
 potentialUser={potentialUser}  
 potentialUsers={potentialUsers}  
 clarificationQuestion={clarificationQuestion}  
 onSelectUser={handleUserSelection}  
 onTryAgain={handleTryAgain}  
 userAssets={userAssets}  
 onAssetSelect={handleAssetSelection}  
 onCreateTicket={createTicket}  
 onRedoProblem={handleRedoProblem}  
 onConfirmUser={handleConfirmation}  
 onListenStart={() => {  
 finalTranscriptRef.current = '';  
 setupSpeechRecognition();  
 if (recognitionRef.current && !isRecognitionActiveRef.current) {  
 recognitionRef.current.start();  
 }  
 }}  
 onListenStop={() => {  
 if (recognitionRef.current && isRecognitionActiveRef.current) {  
 recognitionRef.current.stop();  
 }  
 }}  
 />  
 )}  
 </main>  
  
 <footer className="relative z-10 w-full h-16 px-8 flex justify-end items-center">  
 <button onClick={() => setShowDebug(prev => !prev)} className="p-2 rounded-full bg-white/10 hover:bg-white/20 transition-colors">  
 <DebugIcon className="w-6 h-6 text-white" />  
 </button>  
 </footer>  
 {showDebug && <DebugWindow prompt={debugPrompt} onClose={() => setShowDebug(false)} />}  
 </div>  
 );  
}  
  
// --- UI Display Components ---  
const LiveStatusDisplay = ({ status, interimTranscript, visitorName, iiqUser, problemDescription, troubleshootingSuggestion, identifiedAsset, isListening, errorMessage, potentialUser, potentialUsers, clarificationQuestion, onSelectUser, onTryAgain, userAssets, onAssetSelect, onCreateTicket, onRedoProblem, onConfirmUser, onListenStart, onListenStop }) => {  
 let message = "";  
 if (status === 'initializing') message = "Initializing systems...";  
 if (status === 'awaiting\_scan') message = "Please scan your ID badge, or hold the button below to say your name or ID#.";  
 if (status === 'awaiting\_name') message = "I couldn't locate your record. Please hold the button and say your name or ID# again.";  
 if (status === 'awaiting\_selection') message = "I found a few people. Please tap your name to continue.";  
 if (status === 'awaiting\_asset\_selection') message = "Great. Which device are you having an issue with?";  
 if (status === 'awaiting\_id\_confirmation' && potentialUser) message = `Thanks, ${potentialUser.Name}. Is that correct?`;  
 if (status === 'awaiting\_barcode\_confirmation' && potentialUser) message = `I see you're ${potentialUser.Name}. Is that correct?`;  
 if (status === 'processing') message = "One moment...";  
 if (status === 'verifying') message = "Verifying user...";  
 if (status === 'awaiting\_problem') message = `Thanks, ${visitorName}. Please hold the button and describe your issue.`;  
 if (status === 'awaiting\_clarification') message = clarificationQuestion || "Let me ask a quick follow-up...";  
 if (status === 'awaiting\_confirmation') message = `Please review the details, ${visitorName}. Is this correct?`;  
 if (status === 'error') message = errorMessage || "There was a problem.";  
   
 const nameDisplay = iiqUser && !iiqUser.isFallback   
 ? <><span className="text-cyan-300">{visitorName}</span> <span className="text-sm text-green-400">(Verified)</span></>  
 : <span className="text-yellow-300">{visitorName || '...'}</span>;  
  
 const showTicketDetails = status === 'awaiting\_confirmation';  
 const showUserConfirmationButtons = status === 'awaiting\_id\_confirmation' || status === 'awaiting\_barcode\_confirmation';  
 const showListenButton = ['awaiting\_scan', 'awaiting\_name', 'awaiting\_problem', 'awaiting\_clarification'].includes(status);  
 const showScannerBox = status === 'awaiting\_scan';  
  
 return (  
 <div className="bg-black/60 backdrop-blur-md p-6 rounded-2xl max-w-3xl w-full shadow-2xl border border-gray-500 flex flex-col items-center">  
 <h2 className="text-3xl font-semibold text-cyan-400 mb-4 text-center">{message}</h2>  
   
 {status === 'processing' && <div className="my-4"><LoadingSpinner /></div>}  
   
 {showScannerBox && <ScannerBox />}  
  
 {showListenButton && (  
 <div className="mt-4 flex flex-col items-center">  
 <button   
 onMouseDown={onListenStart}  
 onMouseUp={onListenStop}  
 onTouchStart={onListenStart}  
 onTouchEnd={onListenStop}  
 className={`px-8 py-4 rounded-full transition-all duration-200 flex items-center gap-3 ${isListening ? 'bg-red-600 animate-pulse' : 'bg-cyan-600 hover:bg-cyan-500'}`}  
 >  
 <MicIcon className="w-8 h-8 text-white" />  
 <span className="text-white font-bold text-2xl">{isListening ? 'Listening...' : 'Hold to Speak'}</span>  
 </button>  
 <p className="text-2xl italic text-gray-300 mt-4 min-h-[32px]">"{interimTranscript}"</p>  
 </div>  
 )}  
   
 {errorMessage && status !== 'error' && <p className="text-yellow-300 text-center my-4">{errorMessage}</p>}  
  
 {status === 'awaiting\_selection' && (  
 <>  
 <div className="max-h-64 overflow-y-auto grid grid-cols-2 md:grid-cols-3 gap-3 mt-4 p-2">  
 {potentialUsers.map(user => (  
 <button   
 key={user.UserId}   
 onClick={() => onSelectUser(user)}  
 className="bg-cyan-600/50 hover:bg-cyan-500/80 text-white font-bold py-2 px-3 rounded-lg transition-all duration-200 text-left"  
 >  
 <p className="text-base">{user.Name}</p>  
 <p className="text-xs text-cyan-200">ID: {user.SchoolIdNumber || 'N/A'}</p>  
 </button>  
 ))}  
 </div>  
 <div className="mt-4 text-center">  
 <button onClick={onTryAgain} className="bg-red-600/80 hover:bg-red-500/80 text-white font-bold py-2 px-4 rounded-lg transition-all duration-200">  
 I'm not here. Let's try again.  
 </button>  
 </div>  
 </>  
 )}  
  
 {showUserConfirmationButtons && (  
 <div className="flex justify-center gap-4 pt-4">  
 <button onClick={() => onConfirmUser(true)} className="bg-green-600 hover:bg-green-500 text-white font-bold py-3 px-6 rounded-lg transition-all duration-200">Yes, that's me</button>  
 <button onClick={() => onConfirmUser(false)} className="bg-red-600 hover:bg-red-500 text-white font-bold py-3 px-6 rounded-lg transition-all duration-200">No, that's not me</button>  
 </div>  
 )}  
  
 {status === 'awaiting\_asset\_selection' && (  
 <>  
 <div className="max-h-64 overflow-y-auto grid grid-cols-2 md:grid-cols-3 gap-3 mt-4 p-2">  
 {userAssets.map(asset => (  
 <button   
 key={asset.AssetId}   
 onClick={() => onAssetSelect(asset)}  
 className="bg-cyan-600/50 hover:bg-cyan-500/80 text-white font-bold py-3 px-4 rounded-lg transition-all duration-200 text-left"  
 >  
 <p className="text-base">{asset.Name}</p>  
 <p className="text-xs text-cyan-200">Tag: {asset.AssetTag || 'N/A'}</p>  
 </button>  
 ))}  
 <button   
 onClick={() => onAssetSelect(null)}  
 className="bg-gray-600/50 hover:bg-gray-500/80 text-white font-bold py-3 px-4 rounded-lg transition-all duration-200 text-left"  
 >  
 <p className="text-base">It's something else</p>  
 </button>  
 </div>  
 </>  
 )}  
  
 {showTicketDetails &&   
 <div className="text-left space-y-3 text-xl mt-4">  
 <p><strong className="text-cyan-400">Name:</strong> {nameDisplay}</p>  
 <p><strong className="text-cyan-400">Problem:</strong> <span className="text-white">{problemDescription || '...'}</span></p>  
 {identifiedAsset && <p><strong className="text-cyan-400">Device:</strong> <span className="text-purple-300">{identifiedAsset.Name} ({identifiedAsset.Model?.Name})</span></p>}  
 {troubleshootingSuggestion && (  
 <div className="border-l-4 border-yellow-400 pl-4 mt-3 pt-2 pb-2">  
 <p className="text-yellow-200 text-lg flex items-center gap-2">  
 <SparkleIcon className="w-5 h-5 text-yellow-400 flex-shrink-0" />  
 {troubleshootingSuggestion}  
 </p>  
 </div>  
 )}  
 <div className="flex justify-center gap-4 pt-4">  
 <button onClick={onCreateTicket} className="bg-green-600 hover:bg-green-500 text-white font-bold py-3 px-6 rounded-lg transition-all duration-200">Yes, Create Ticket</button>  
 <button onClick={onRedoProblem} className="bg-red-600 hover:bg-red-500 text-white font-bold py-3 px-6 rounded-lg transition-all duration-200">No, Let's Try Again</button>  
 </div>  
 </div>  
 }  
 </div>  
 );  
};  
  
const ConfirmationDisplay = ({ ticket }) => (  
 <div className="bg-teal-900/80 backdrop-blur-md p-8 rounded-2xl max-w-3xl w-full shadow-2xl border-2 border-cyan-500">  
 <CheckCircleIcon className="w-20 h-20 mx-auto text-cyan-400" />  
 <h2 className="text-4xl font-bold mt-4">Ticket Created!</h2>  
 <p className="text-lg mt-2">A technician will be with you shortly. This screen will reset automatically.</p>  
 <div className="mt-6 bg-black/40 p-4 rounded-lg text-left text-xl space-y-2">  
 <p><strong>Ticket #:</strong> <span className="font-mono">{ticket.ticketNumber}</span></p>  
 <p><strong>Title:</strong> {ticket.title}</p>  
 <p><strong>Visitor:</strong> {ticket.visitorName}</p>  
 </div>  
 </div>  
);  
  
const DebugWindow = ({ prompt, onClose }) => (  
 <div className="absolute inset-0 bg-black/80 backdrop-blur-sm z-50 flex items-center justify-center p-4">  
 <div className="bg-gray-900 text-white rounded-lg shadow-2xl max-w-4xl w-full max-h-full flex flex-col">  
 <header className="p-4 border-b border-gray-700 flex justify-between items-center">  
 <h3 className="text-lg font-semibold">AI Debug Prompt</h3>  
 <button onClick={onClose} className="text-gray-400 hover:text-white">×</button>  
 </header>  
 <pre className="p-4 overflow-auto text-sm whitespace-pre-wrap">  
 {prompt}  
 </pre>  
 </div>  
 </div>  
);