```
import React, { useState, useEffect, useRef, createContext, useContext
} from 'react';
import { initializeApp } from "firebase/app";
import { getAuth, signInAnonymously, onAuthStateChanged } from
"firebase/auth";
import { getFirestore, collection, addDoc, onSnapshot, query, orderBy,
limit } from "firebase/firestore";

// --- PASTE YOUR FIREBASE CONFIG HERE ---
const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_AUTH_DOMAIN",
  projectId: "YOUR_PROJECT_ID",
  storageBucket: "YOUR_STORAGE_BUCKET",
  messagingSenderId: "YOUR_MESSAGING_SENDER_ID",
  appId: "YOUR_APP_ID"
};
// ----------------------------------------

// --- Firebase Initialization ---
const AppContext = createContext();
const appId = typeof __app_id !== 'undefined' ? __app_id :
'ai-wellness-companion';
const effectiveFirebaseConfig = typeof __firebase_config !==
'undefined' ? JSON.parse(__firebase_config) : firebaseConfig;

const app = initializeApp(effectiveFirebaseConfig);
const auth = getAuth(app);
const db = getFirestore(app);

// --- Gemini API Helper ---
const callGeminiApi = async (prompt, systemInstruction) => {
    const apiKey = ""; // Canvas will provide this
    const apiUrl =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-fl
ash-preview-05-20:generateContent?key=${apiKey}`;

    const payload = {
        contents: [{ parts: [{ text: prompt }] }],
        systemInstruction: {
            parts: [{ text: systemInstruction }]
        },
    };

    // Exponential backoff for retries
    let response;
    for (let i = 0; i < 4; i++) {
        try {
```

```
            response = await fetch(apiUrl, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(payload)
            });
            if (response.ok) break;
        } catch (error) {
            console.error('API call failed:', error);
        }
        await new Promise(resolve => setTimeout(resolve, Math.pow(2,
i) * 1000));
    }

    if (!response || !response.ok) {
        return "Sorry, I'm having trouble connecting right now. Please
try again later.";
    }

    try {
        const result = await response.json();
        return result.candidates?.[0]?.content?.parts?.[0]?.text ||
"I'm not sure how to respond to that. Could you rephrase?";
    } catch (error) {
        console.error("Error parsing Gemini response:", error);
        return "There was an error processing the response.";
    }
};


// --- Dynamically load external scripts ---
const loadScript = (src) => {
  return new Promise((resolve, reject) => {
    if (document.querySelector(`script[src="${src}"]`)) {
        return resolve();
    }
    const script = document.createElement('script');
    script.src = src;
    script.onload = () => resolve();
    script.onerror = () => reject(new Error(`Script load error for
${src}`));
    document.body.appendChild(script);
  });
};


// --- Main App Component ---
export default function App() {
  const [user, setUser] = useState(null);
```

```
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, async (currentUser)
=> {
      if (currentUser) {
        setUser(currentUser);
      } else {
        try {
          await signInAnonymously(auth);
        } catch (error) {
          console.error("Error signing in anonymously:", error);
        }
      }
      setLoading(false);
    });
    return () => unsubscribe();
  }, []);

  if (loading) {
    return <LoadingScreen />;
  }

  return (
    <AppContext.Provider value={{ user, db, appId }}>
      <div className="bg-gray-900 text-white min-h-screen font-sans">
        <Header />
        <main className="container mx-auto px-4 py-8">
          <Dashboard />
        </main>
        <Footer />
      </div>
    </AppContext.Provider>
  );
}


// --- UI Components ---

function LoadingScreen() {
  return (
    <div className="flex items-center justify-center min-h-screen
bg-gray-900">
      <div className="text-center">
        <svg className="animate-spin h-10 w-10 text-cyan-400 mx-auto
mb-4" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24
24">
          <circle className="opacity-25" cx="12" cy="12" r="10"
```

```
stroke="currentColor" strokeWidth="4"></circle>
          <path className="opacity-75" fill="currentColor" d="M4 12a8
8-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042
1.135 5.824 3 7.938l3-2.647z"></path>
        </svg>
        <h1 className="text-2xl font-semibold text-white">Connecting
to your space...</h1>
        <p className="text-gray-400">Please wait a moment.</p>
      </div>
    </div>
  );
}

function Header() {
  const { user } = useContext(AppContext);
  return (
    <header className="bg-gray-900/80 backdrop-blur-sm sticky top-0
z-50 border-b border-cyan-500/30">
      <nav className="container mx-auto px-4 py-3 flex justify-between
items-center">
        <div className="flex items-center space-x-3">
           <svg className="w-8 h-8 text-cyan-400" fill="none"
stroke="currentColor" viewBox="0 0 24 24"
xmlns="http://www.w3.org/2000/svg"><path strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M12 3v1m0 16v1m9-9h-1M4
12H3m15.364 6.364l-.707-.707M6.343 6.343l-.707-.707m12.728
0l-.707.707M6.343 17.657l-.707.707M16 12a4 4 0 11-8 0 4 4 0 018
0z"></path></svg>
          <h1 className="text-xl font-bold text-white
tracking-wider">Youth Mental Wellness</h1>
        </div>
        <div>
          {user && <span className="text-sm text-gray-400 hidden
sm:block">UID: {user.uid}</span>}
        </div>
      </nav>
    </header>
  );
}

function Dashboard() {
  const [autoDetectedMood, setAutoDetectedMood] = useState(null);

  return (
    <div className="grid grid-cols-1 lg:grid-cols-3 gap-8">
      <div className="lg:col-span-2 space-y-8">
        <FacialMoodDetector onMoodDetected={setAutoDetectedMood} />
        <MoodCheckIn autoDetectedMood={autoDetectedMood} />
```

```
        <Journaling />
      </div>
      <div className="lg:col-span-1">
        <AIChat />
      </div>
    </div>
  );
}

function FacialMoodDetector({ onMoodDetected }) {
    const videoRef = useRef(null);
    const canvasRef = useRef(null);
    const [modelsLoaded, setModelsLoaded] = useState(false);
    const [detectionStatus, setDetectionStatus] =
useState('Initializing');
    const [detectedMood, setDetectedMood] = useState(null);
    const [isCameraOn, setIsCameraOn] = useState(false);
    const intervalRef = useRef(null);

    useEffect(() => {
        const loadModels = async () => {
            const MODEL_URL =
'https://cdn.jsdelivr.net/npm/@vladmandic/face-api/model/';
            try {
                await
loadScript('https://cdn.jsdelivr.net/npm/face-api.js@0.22.2/dist/face-
api.min.js');
                await Promise.all([

faceapi.nets.tinyFaceDetector.loadFromUri(MODEL_URL),

faceapi.nets.faceLandmark68Net.loadFromUri(MODEL_URL),

faceapi.nets.faceRecognitionNet.loadFromUri(MODEL_URL),

faceapi.nets.faceExpressionNet.loadFromUri(MODEL_URL),
                ]);
                setModelsLoaded(true);
                setDetectionStatus('Ready to start');
            } catch (error) {
                console.error("Failed to load face-api models:",
error);
                setDetectionStatus('Error loading AI models.');
            }
        };
        loadModels();

        return () => {
```

```
            if(intervalRef.current)
clearInterval(intervalRef.current);
            if(videoRef.current && videoRef.current.srcObject) {
                videoRef.current.srcObject.getTracks().forEach(track
=> track.stop());
            }
        }
    }, []);

    const startVideo = () => {
        navigator.mediaDevices.getUserMedia({ video: {} })
            .then(stream => {
                if (videoRef.current) {
                    videoRef.current.srcObject = stream;
                    setIsCameraOn(true);
                    setDetectionStatus('Detecting...');
                }
            })
            .catch(err => {
                console.error("error:", err);
                setDetectionStatus('Camera permission denied.');
            });
    };

    const handleVideoPlay = () => {
        intervalRef.current = setInterval(async () => {
            if (videoRef.current && !videoRef.current.paused && typeof
faceapi !== 'undefined') {
                const detections = await
faceapi.detectAllFaces(videoRef.current, new
faceapi.TinyFaceDetectorOptions()).withFaceLandmarks().withFaceExpress
ions();
                if (detections.length > 0) {
                    const expressions = detections[0].expressions;
                    const primaryMood =
Object.keys(expressions).reduce((a, b) => expressions[a] >
expressions[b] ? a : b);
                    const moodMapping = { 'happy': 'Joyful', 'sad':
'Sad', 'angry': 'Angry', 'neutral': 'Content', 'surprised': 'Anxious'
};

                    if (moodMapping[primaryMood] && detectedMood !==
moodMapping[primaryMood]) {
                        setDetectedMood(moodMapping[primaryMood]);
                        onMoodDetected(moodMapping[primaryMood]);
                    }
                }
            }
```

```
        }, 700);
    };

    return (
        <section className="bg-gray-800/50 p-6 rounded-2xl shadow-lg
border border-gray-700">
            <h2 className="text-xl font-semibold mb-4
text-cyan-300">Automatic Mood Detection</h2>
            <div className="flex flex-col items-center">
                 <div ref={videoRef} onPlay={handleVideoPlay}
autoPlay muted playsInline className="w-full h-full object-cover" />
                    <canvas ref={canvasRef} className="absolute top-0
left-0" />
                </div>
                {!isCameraOn ? (
                     <button onClick={startVideo}
disabled={!modelsLoaded} className="bg-cyan-600 hover:bg-cyan-500
text-white font-bold py-2 px-4 rounded-lg transition-colors
disabled:bg-gray-500">
                        {modelsLoaded ? 'Start Camera' : 'Loading AI
Models...'}
                    </button>
                ) : (
                    <p className="text-lg">Detected Mood: <span
className="font-bold text-cyan-400">{detectedMood || '...'}</span></p>
                )}
                 <p className="text-sm text-gray-400
mt-2">{detectionStatus}</p>
            </div>
        </section>
    );
}


function MoodCheckIn({ autoDetectedMood }) {
  const { user, db, appId } = useContext(AppContext);
  const [selectedMood, setSelectedMood] = useState(null);
  const [isSubmitted, setIsSubmitted] = useState(false);
  const moods = [
    { name: 'Joyful', icon: '😊' }, { name: 'Calm', icon: '😌' },
    { name: 'Sad', icon: '😢' }, { name: 'Anxious', icon: '😟' },
    { name: 'Angry', icon: '😠' }, { name: 'Content', icon: '🙂' },
  ];

  const handleMoodSelect = async (moodName) => {
```

```
    const mood = moods.find(m => m.name === moodName);
    if (!user || !mood || isSubmitted) return;

    setSelectedMood(mood);
    setIsSubmitted(true);
    try {
        const privateCollectionPath =
`artifacts/${appId}/users/${user.uid}/moods`;
        await addDoc(collection(db, privateCollectionPath), {
            mood: mood.name,
            icon: mood.icon,
            timestamp: new Date(),
        });
    } catch (error) {
        console.error("Error adding mood: ", error);
    }

    setTimeout(() => {
      setIsSubmitted(false);
      setSelectedMood(null);
    }, 2500);
  };

  useEffect(() => {
      if(autoDetectedMood) {
          handleMoodSelect(autoDetectedMood);
      }
  }, [autoDetectedMood]);

  return (
    <section className="bg-gray-800/50 p-6 rounded-2xl shadow-lg
border border-gray-700">
      <h2 className="text-xl font-semibold mb-4 text-cyan-300">How are
you feeling right now?</h2>
      {isSubmitted ? (
        <div className="text-center py-8 transition-opacity
duration-300">
          <p className="text-3xl">{selectedMood?.icon}</p>
          <p className="mt-2 text-lg">Thanks for sharing. Your mood
'{selectedMood?.name}' has been logged.</p>
        </div>
      ) : (
        <div className="grid grid-cols-3 sm:grid-cols-6 gap-4">
          {moods.map((mood) => (
            <button
              key={mood.name}
              onClick={() => handleMoodSelect(mood.name)}
              className={`flex flex-col items-center justify-center
```

```
p-4 rounded-xl transition-all duration-300 ease-in-out transform
hover:-translate-y-1 hover:shadow-2xl focus:outline-none focus:ring-2
focus:ring-offset-2 focus:ring-offset-gray-800 focus:ring-cyan-400
bg-gray-700 hover:bg-gray-600`}
            >
              <span className="text-4xl">{mood.icon}</span>
              <span className="mt-2 text-sm
font-medium">{mood.name}</span>
            </button>
          ))}
        </div>
      )}
    </section>
  );
}

function Journaling() {
    const { user, db, appId } = useContext(AppContext);
    const [entry, setEntry] = useState('');
    const [entries, setEntries] = useState([]);
    const [isSaving, setIsSaving] = useState(false);
    const [insight, setInsight] = useState('');
    const [isGeneratingInsight, setIsGeneratingInsight] =
useState(false);

    useEffect(() => {
        if (!user) return;
        const privateCollectionPath =
`artifacts/${appId}/users/${user.uid}/journal`;
        const q = query(collection(db, privateCollectionPath),
orderBy('timestamp', 'desc'));

        const unsubscribe = onSnapshot(q, (querySnapshot) => {
            const journalEntries = querySnapshot.docs.map(doc => ({
id: doc.id, ...doc.data() }));
            setEntries(journalEntries);
        }, (error) => console.error("Error fetching journal entries:
", error));

        return () => unsubscribe();
    }, [user, db, appId]);

    const handleSaveEntry = async () => {
        if (entry.trim() === '' || !user) return;
        setIsSaving(true);
        try {
            const privateCollectionPath =
`artifacts/${appId}/users/${user.uid}/journal`;
```

```
            await addDoc(collection(db, privateCollectionPath), {
text: entry, timestamp: new Date() });
            setEntry('');
            setInsight(''); // Clear previous insight on new entry
        } catch (error) {
            console.error("Error saving journal entry: ", error);
        } finally {
            setIsSaving(false);
        }
    };

    const handleGetInsights = async () => {
        if (entry.trim() === '') return;
        setIsGeneratingInsight(true);
        setInsight('');
        const systemPrompt = "You are an empathetic wellness coach.
Read the following journal entry from a user. Provide a short (2-3
sentences), compassionate, and insightful reflection. Focus on
validating their feelings and offering a gentle, encouraging
perspective. Do not give medical advice. Frame your response as if you
are speaking directly to the user.";
        const response = await callGeminiApi(entry, systemPrompt);
        setInsight(response);
        setIsGeneratingInsight(false);
    };

    return (
        <section className="bg-gray-800/50 p-6 rounded-2xl shadow-lg
border border-gray-700">
            <h2 className="text-xl font-semibold mb-4
text-cyan-300">Daily Journal</h2>
            <textarea
                className="w-full h-32 p-3 bg-gray-700 rounded-lg
border border-gray-600 focus:outline-none focus:ring-2
focus:ring-cyan-400 text-white transition"
                placeholder="What's on your mind?"
                value={entry}
                onChange={(e) => setEntry(e.target.value)}
            ></textarea>
            <div className="mt-4 flex space-x-4">
                <button
                    onClick={handleSaveEntry}
                    disabled={isSaving || entry.trim() === ''}
                    className="flex-1 bg-cyan-600 hover:bg-cyan-500
text-white font-bold py-2 px-4 rounded-lg transition-colors
disabled:bg-gray-500 disabled:cursor-not-allowed"
                >
                    {isSaving ? 'Saving...' : 'Save Entry'}
```

```
                </button>
                <button
                    onClick={handleGetInsights}
                    disabled={isGeneratingInsight || entry.trim() ===
''}
                    className="flex-1 bg-cyan-700 hover:bg-cyan-600
text-white font-bold py-2 px-4 rounded-lg transition-colors
disabled:bg-gray-500 disabled:cursor-not-allowed"
                >
                   {isGeneratingInsight ? 'Thinking...' : '✨ Get
Insight'}
                </button>
            </div>

            {isGeneratingInsight && <p className="text-center
text-gray-400 mt-4">Generating your insight...</p>}
            {insight && (
                <div className="mt-6 p-4 bg-gray-700/50 border-l-4
border-cyan-400 rounded-r-lg">
                    <h4 className="font-semibold
text-cyan-300">Wellness Insight</h4>
                    <p className="text-gray-300 mt-2">{insight}</p>
                </div>
            )}

            <div className="mt-6">
                <h3 className="text-lg font-semibold mb-2">Recent
Entries</h3>
                <div className="space-y-4 max-h-60 overflow-y-auto
pr-2">
                    {entries.length > 0 ? entries.map(e => (
                        <div key={e.id} className="bg-gray-700 p-3
rounded-lg">
                            <p className="text-sm text-gray-300
whitespace-pre-wrap">{e.text}</p>
                            <p className="text-xs text-gray-500 mt-2
text-right">

{e.timestamp?.toDate().toLocaleDateString()}
                            </p>
                        </div>
                    )) : <p className="text-gray-400">No entries yet.
Write your first one above!</p>}
                </div>
            </div>
        </section>
    );
}
```

```
function AIChat() {
    const { user, db, appId } = useContext(AppContext);
    const [messages, setMessages] = useState([]);
    const [input, setInput] = useState('');
    const [isListening, setIsListening] = useState(false);
    const [isInitializing, setIsInitializing] = useState(true);
    const [isAwaitingResponse, setIsAwaitingResponse] =
useState(false);
    const [latestMood, setLatestMood] = useState(null);
    const [latestJournal, setLatestJournal] = useState(null);
    const chatEndRef = useRef(null);
    const recognitionRef = useRef(null);

    // Effect for fetching data
    useEffect(() => {
        if (!user) return;

        const moodQuery = query(collection(db,
`artifacts/${appId}/users/${user.uid}/moods`), orderBy('timestamp',
'desc'), limit(1));
        const unsubMood = onSnapshot(moodQuery, (snapshot) => {
            setLatestMood(snapshot.docs[0]?.data() || 'new_user');
        }, (error) => {
            console.error("Error fetching mood:", error);
            setLatestMood('new_user');
        });

        const journalQuery = query(collection(db,
`artifacts/${appId}/users/${user.uid}/journal`), orderBy('timestamp',
'desc'), limit(1));
        const unsubJournal = onSnapshot(journalQuery, (snapshot) => {
            setLatestJournal(snapshot.docs[0]?.data() || 'new_user');
        }, (error) => {
            console.error("Error fetching journal:", error);
            setLatestJournal('new_user');
        });

        return () => {
            unsubMood();
            unsubJournal();
        };
    }, [user, db, appId]);

    // Effect for updating initial message based on fetched data
    useEffect(() => {
        if (latestMood && latestJournal) {
            let initialMessage = "Hello! I'm your personal wellness
```

```
companion. How can I support you today?";
            if (latestMood !== 'new_user') {
                initialMessage = `I notice you're feeling
${latestMood.mood.toLowerCase()} today. Is there anything on your
mind?`;
            }
            if (latestMood !== 'new_user' && latestJournal !==
'new_user') {
                const journalSnippet = latestJournal.text.substring(0,
40);
                initialMessage = `I sense you're feeling
${latestMood.mood.toLowerCase()}. I also saw your recent journal entry
about "${journalSnippet}...". Would you like to talk about it?`;
            }
            setMessages([{ role: "model", parts: [{ text:
initialMessage }] }]);
            setIsInitializing(false);
        }
    }, [latestMood, latestJournal]);

    useEffect(() => {
        const SpeechRecognition = window.SpeechRecognition ||
window.webkitSpeechRecognition;
        if (SpeechRecognition) {
            const recognition = new SpeechRecognition();
            recognition.continuous = false;
            recognition.interimResults = false;
            recognition.lang = 'en-US';
            recognition.onresult = (event) => {
                const transcript = event.results[0][0].transcript;
                setIsListening(false);
                if (transcript.trim() !== '') handleSend(transcript);
            };
            recognition.onerror = (event) => {
                console.error('Speech recognition error:',
event.error);
                setIsListening(false);
            };
            recognition.onend = () => setIsListening(false);
            recognitionRef.current = recognition;
        }
    }, []);

    const toggleListening = () => {
        if (isListening) {
            recognitionRef.current?.stop();
        } else if (recognitionRef.current) {
            recognitionRef.current.start();
```

```
            setIsListening(true);
        }
    };

    useEffect(() => {
        chatEndRef.current?.scrollIntoView({ behavior: "smooth" });
    }, [messages]);

    const handleSend = async (textToSend) => {
        const currentInput = typeof textToSend === 'string' ?
textToSend : input;
        if (currentInput.trim() === '' || isAwaitingResponse) return;

        const newUserMessage = { role: "user", parts: [{ text:
currentInput }] };
        const updatedMessages = [...messages, newUserMessage];
        setMessages(updatedMessages);
        setInput('');
        setIsAwaitingResponse(true);

        const systemPrompt = "You are an AI wellness companion for
youth. Your personality is empathetic, patient, and supportive. You
are not a doctor and must never give medical advice. Keep your
responses concise and conversational. Your goal is to help the user
explore their feelings and feel heard.";
        // We will create a prompt from the conversation history
        const promptFromHistory = updatedMessages.map(msg =>
`${msg.role === 'model' ? 'AI' : 'User'}:
${msg.parts[0].text}`).join('\n');

        const aiResponseText = await callGeminiApi(promptFromHistory,
systemPrompt);

        const newAiMessage = { role: "model", parts: [{ text:
aiResponseText }] };
        setMessages(prev => [...prev, newAiMessage]);
        setIsAwaitingResponse(false);
    };

    const handleTextSend = () => handleSend(input);

    return (
        <section className="bg-gray-800/50 rounded-2xl shadow-lg
border border-gray-700 flex flex-col h-[85vh]">
            <h2 className="text-xl font-semibold p-4 border-b
border-gray-700 text-cyan-300">✨ AI Companion</h2>
            <div className="flex-1 p-4 space-y-4 overflow-y-auto">
                {isInitializing ? ( <div className="flex
```

```
justify-center items-center h-full"><p
className="text-gray-400">Personalizing your chat...</p></div> ) : (
                    messages.map((msg, index) => (
                        <div key={index} className={`flex ${msg.role
=== 'user' ? 'justify-end' : 'justify-start'}`}>
                            <div className={`max-w-xs lg:max-w-md px-4
py-2 rounded-2xl shadow-md ${msg.role === 'user' ? 'bg-cyan-600
text-white rounded-br-none' : 'bg-gray-600 text-white
rounded-bl-none'}`}>
                                {msg.parts[0].text}
                            </div>
                        </div>
                    ))
                )}
                {isAwaitingResponse && <div className="flex
justify-start"><div className="max-w-xs lg:max-w-md px-4 py-2
rounded-2xl shadow-md bg-gray-600 text-white
rounded-bl-none">Typing...</div></div>}
                <div ref={chatEndRef} />
            </div>
            <div className="p-4 border-t border-gray-700 flex
items-center">
                 <button onClick={toggleListening} className={`p-2
rounded-full transition-colors ${isListening ? 'bg-red-500
animate-pulse' : 'bg-cyan-600 hover:bg-cyan-500'}`}
disabled={isAwaitingResponse}>
                    <svg className="w-6 h-6 text-white" fill="none"
stroke="currentColor" viewBox="0 0 24 24"
xmlns="http://www.w3.org/2000/svg"><path strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M19 11a7 7 0 01-7 7m0 0a7 7
0 01-7-7m7 7v4m0 0H8m4 0h4m-4-8a3 3 0 01-3-3V5a3 3 0 116 0v6a3 3 0
01-3 3z" /></svg>
                </button>
                <input
                    type="text"
                    value={input}
                    onChange={(e) => setInput(e.target.value)}
                    onKeyPress={(e) => e.key === 'Enter' &&
handleTextSend()}
                    className="w-full bg-gray-700 rounded-full py-2
px-4 focus:outline-none focus:ring-2 focus:ring-cyan-400 text-white
transition mx-2"
                    placeholder="Type or speak..."
                    disabled={isInitializing || isAwaitingResponse}
                />
                <button onClick={handleTextSend} className="p-2
bg-cyan-600 rounded-full hover:bg-cyan-500 transition-colors
disabled:bg-gray-500" disabled={isInitializing || isAwaitingResponse}>
```

```
                    <svg className="w-6 h-6 text-white" fill="none"
stroke="currentColor" viewBox="0 0 24 24"
xmlns="http://www.w3.org/2000/svg"><path strokeLinecap="round"
strokeLinejoin="round" strokeWidth={2} d="M5 10l7-7m0 0l7 7m-7-7v18"
/></svg>
                </button>
            </div>
        </section>
    );
}

function Footer() {
  return (
    <footer className="bg-gray-900 border-t border-cyan-500/30 mt-8">
      <div className="container mx-auto px-4 py-4 text-center
text-gray-500">
        <p>© 2025 Youth Mental Wellness. All rights reserved.</p>
        <p className="text-xs mt-1">This application is for
demonstration purposes and is not a substitute for professional
medical advice.</p>
      </div>
    </footer>
  );
}
```