import React, { useState, useEffect, useRef, useCallback } from 'react';

// Utility function to generate a single ECG point (Y-value) at a given time for a specific type

// This function simulates the different phases of an ECG waveform (P, QRS, T) and adds noise.

// For MI, it introduces simulated ST elevation, deeper Q waves, and inverted T waves.

const generateECGPoint = (time, type) => {

  // Simulate a heart rate of 60 bpm (1 beat per second)

  const beatDuration = 1000; // milliseconds for one complete ECG cycle

  const timeInBeat = time % beatDuration; // Current time within the 1-second beat cycle

  let y = 0; // Baseline for the ECG signal

  // Simulate P wave (at the beginning of the beat cycle)

  if (timeInBeat >= 0 && timeInBeat < 100) {

    y += Math.sin(timeInBeat / 100 \* Math.PI) \* 10; // A small positive hump

  }

  // Simulate PR segment (flat line after P wave)

  else if (timeInBeat >= 100 && timeInBeat < 150) {

    y += 0;

  }

  // Simulate QRS complex (the main sharp spike)

  else if (timeInBeat >= 150 && timeInBeat < 250) {

    if (timeInBeat < 170) { // Q wave (downward deflection)

      y += (timeInBeat - 170) / 20 \* 30;

    } else if (timeInBeat < 200) { // R wave (large upward deflection)

      y += ((timeInBeat - 170) / 30) \* -80;

    } else { // S wave (downward deflection after R)

      y += (timeInBeat - 200) / 50 \* 50;

    }

  }

  // Simulate ST segment (flat line after S wave, before T wave)

  else if (timeInBeat >= 250 && timeInBeat < 320) {

    y += 0;

  }

  // Simulate T wave (a broader hump after ST segment)

  else if (timeInBeat >= 320 && timeInBeat < 450) {

    y += Math.sin((timeInBeat - 320) / 130 \* Math.PI) \* 20;

  }

  // Add random noise to make the signal less perfect

  y += (Math.random() - 0.5) \* 5;

  // Apply Myocardial Infarction (MI) specific changes if `type` is 'mi'

  if (type === 'mi') {

    // Simulate ST elevation: A significant upward shift of the ST segment, characteristic of STEMI

    if (timeInBeat >= 250 && timeInBeat < 320) {

      y -= 30; // Elevate ST segment by a fixed amount

    }

    // Simulate pathological Q wave: A deeper and wider Q wave, often indicating past MI

    if (timeInBeat >= 150 && timeInBeat < 170) {

      y += (timeInBeat - 170) / 20 \* 60; // Make Q wave significantly deeper

    }

    // Simulate T wave inversion: The T wave goes below the baseline, common in MI

    if (timeInBeat >= 320 && timeInBeat < 450) {

      y \*= -1.5; // Invert and amplify the T wave

    }

  }

  return y; // Return the calculated Y-value for the ECG point

};

// Main App Component

function App() {

  const [ecgType, setEcgType] = useState(null); // 'normal' or 'mi' for stream type

  const [diagnosis, setDiagnosis] = useState('');

  const [confidenceScore, setConfidenceScore] = useState(0);

  const [ecgCharacteristics, setEcgCharacteristics] = useState([]);

  const [ecgData, setEcgData] = useState([]); // Array to store real-time ECG points

  const [isStreaming, setIsStreaming] = useState(false); // Controls data stream

  const animationFrameRef = useRef(null); // Ref for requestAnimationFrame ID

  const lastTimeRef = useRef(0); // Ref to keep track of last animation time

  const ecgSimTimeRef = useRef(0); // Ref to keep track of simulated ECG time

  // Max number of points to display on the ECG canvas for scrolling effect

  const MAX\_ECG\_POINTS = 600; // Corresponds to canvas width for a 1:1 pixel mapping

  // Function to simulate AI analysis based on the selected ECG type

  const performAIAnalysis = useCallback((type) => {

    let newDiagnosis = '';

    let newConfidenceScore = 0;

    let newCharacteristics = [];

    if (type === 'normal') {

      newDiagnosis = 'Normal Sinus Rhythm';

      newConfidenceScore = Math.floor(Math.random() \* (99 - 90 + 1)) + 90; // 90-99%

      newCharacteristics = [

        'Regular rhythm',

        'Heart rate 60-100 bpm',

        'Normal P waves preceding each QRS complex',

        'Normal PR interval (0.12-0.20s)',

        'Normal QRS duration (<0.12s)',

        'Isoelectric ST segment',

        'Upright T waves',

      ];

    } else if (type === 'mi') {

      newDiagnosis = 'Myocardial Infarction (Simulated)';

      newConfidenceScore = Math.floor(Math.random() \* (95 - 85 + 1)) + 85; // 85-95%

      newCharacteristics = [

        'ST segment elevation or depression (depending on MI type)',

        'Pathological Q waves (wider and deeper than normal)',

        'T-wave inversion or hyperacute T waves',

        'Possible abnormal R-wave progression',

        'May be associated with arrhythmias',

      ];

    }

    setDiagnosis(newDiagnosis);

    setConfidenceScore(newConfidenceScore);

    setEcgCharacteristics(newCharacteristics);

  }, []);

  // Effect to manage the real-time ECG data stream

  useEffect(() => {

    if (isStreaming && ecgType) {

      lastTimeRef.current = performance.now(); // Initialize last time for animation

      ecgSimTimeRef.current = 0; // Reset simulated ECG time

      const animate = (currentTime) => {

        const deltaTime = currentTime - lastTimeRef.current;

        lastTimeRef.current = currentTime;

        // Generate new ECG points based on deltaTime

        const pointsToGenerate = Math.floor(deltaTime / 5); // Generate a point every 5ms

        for (let i = 0; i < pointsToGenerate; i++) {

          const newPoint = generateECGPoint(ecgSimTimeRef.current, ecgType);

          setEcgData(prevData => {

            const updatedData = [...prevData, newPoint];

            // Keep only the latest MAX\_ECG\_POINTS for a scrolling effect

            return updatedData.slice(Math.max(updatedData.length - MAX\_ECG\_POINTS, 0));

          });

          ecgSimTimeRef.current += 5; // Advance simulated ECG time

        }

        animationFrameRef.current = requestAnimationFrame(animate);

      };

      animationFrameRef.current = requestAnimationFrame(animate);

      // Cleanup function to stop the animation frame when component unmounts or stream stops

      return () => {

        if (animationFrameRef.current) {

          cancelAnimationFrame(animationFrameRef.current);

        }

      };

    } else {

      // If not streaming, ensure no animation frame is pending

      if (animationFrameRef.current) {

        cancelAnimationFrame(animationFrameRef.current);

      }

      setEcgData([]); // Clear data when stream stops

      setDiagnosis(''); // Clear diagnosis

      setConfidenceScore(0); // Clear confidence score

      setEcgCharacteristics([]); // Clear characteristics

    }

  }, [isStreaming, ecgType, performAIAnalysis]);

  // Handler for "Start Normal ECG Stream" button

  const startNormalStream = () => {

    setEcgType('normal');

    setIsStreaming(true);

    performAIAnalysis('normal'); // Perform analysis when stream type is set

  };

  // Handler for "Start MI ECG Stream" button

  const startMIStream = () => {

    setEcgType('mi');

    setIsStreaming(true);

    performAIAnalysis('mi'); // Perform analysis when stream type is set

  };

  // Handler for "Stop Stream" button

  const stopStream = () => {

    setIsStreaming(false);

    setEcgType(null); // Reset ECG type

  };

  return (

    <div className="min-h-screen bg-gray-100 flex flex-col items-center justify-center p-4 font-inter">

      <div className="bg-white rounded-lg shadow-xl p-8 w-full max-w-4xl">

        <h1 className="text-3xl font-bold text-gray-800 mb-6 text-center">

          Real-Time AI-Based ECG Analyzer (Simulated)

        </h1>

        <p className="text-gray-600 mb-8 text-center">

          This application simulates a real-time ECG stream and an AI's ability to differentiate between a normal ECG and one indicative of Myocardial Infarction.

          Select a scenario to start the live stream and see a mock diagnosis.

        </p>

        {/\* Action Buttons \*/}

        <div className="flex flex-col sm:flex-row justify-center gap-4 mb-10">

          <button

            onClick={startNormalStream}

            className="bg-green-600 hover:bg-green-700 text-white font-semibold py-3 px-6 rounded-lg shadow-md transition duration-300 ease-in-out transform hover:scale-105"

            disabled={isStreaming} // Disable buttons when streaming

          >

            Start Normal ECG Stream

          </button>

          <button

            onClick={startMIStream}

            className="bg-red-600 hover:bg-red-700 text-white font-semibold py-3 px-6 rounded-lg shadow-md transition duration-300 ease-in-out transform hover:scale-105"

            disabled={isStreaming} // Disable buttons when streaming

          >

            Start MI ECG Stream

          </button>

          {isStreaming && (

            <button

              onClick={stopStream}

              className="bg-gray-500 hover:bg-gray-600 text-white font-semibold py-3 px-6 rounded-lg shadow-md transition duration-300 ease-in-out transform hover:scale-105"

            >

              Stop Stream

            </button>

          )}

        </div>

        {/\* Real-time ECG Display \*/}

        <ECGDisplay ecgData={ecgData} />

        {/\* Analysis Results \*/}

        {diagnosis && (

          <div className="mt-8 border-t border-gray-200 pt-8">

            <h2 className="text-2xl font-semibold text-gray-700 mb-4 text-center">

              AI Analysis Result:

            </h2>

            <div className={`p-6 rounded-lg shadow-inner ${ecgType === 'normal' ? 'bg-green-50' : 'bg-red-50'}`}>

              <p className="text-lg mb-2">

                <span className="font-medium">Diagnosis:</span>{' '}

                <span className={`font-bold ${ecgType === 'normal' ? 'text-green-800' : 'text-red-800'}`}>

                  {diagnosis}

                </span>

              </p>

              <p className="text-lg mb-4">

                <span className="font-medium">Confidence Score:</span>{' '}

                <span className="font-bold text-gray-700">{confidenceScore}%</span>

              </p>

              {/\* ECG Characteristics \*/}

              <div className="mt-6">

                <h3 className="text-xl font-semibold text-gray-700 mb-3">

                  Typical ECG Characteristics:

                </h3>

                <ul className="list-disc list-inside text-gray-700 space-y-1">

                  {ecgCharacteristics.map((char, index) => (

                    <li key={index}>{char}</li>

                  ))}

                </ul>

                <p className="text-sm text-gray-500 mt-4 italic">

                  Note: This application provides a simulated real-time ECG stream and AI analysis for educational purposes only. It is not a medical device and should not be used for actual diagnosis.

                </p>

              </div>

            </div>

          </div>

        )}

      </div>

    </div>

  );

}

// ECG Display Component using Canvas for real-time plotting

function ECGDisplay({ ecgData }) {

  const canvasRef = useRef(null); // Ref to access the canvas DOM element

  // Effect hook to draw the ECG waveform whenever `ecgData` changes

  useEffect(() => {

    const canvas = canvasRef.current;

    if (!canvas) return; // Exit if canvas element is not available

    const ctx = canvas.getContext('2d'); // Get 2D rendering context

    if (!ctx) return; // Exit if context is not available

    // Make canvas responsive to its container

    // Get the actual width and height from the computed style

    const containerWidth = canvas.offsetWidth;

    const containerHeight = canvas.offsetHeight;

    canvas.width = containerWidth;

    canvas.height = containerHeight;

    ctx.clearRect(0, 0, canvas.width, canvas.height); // Clear the entire canvas

    // Define ECG grid parameters

    const majorGridColor = '#e0e0e0'; // Light grey for major lines

    const minorGridColor = '#f0f0f0'; // Lighter grey for minor lines

    const majorGridSpacing = 50; // Pixels per major grid square (e.g., 5mm at 10px/mm)

    const minorGridSpacing = majorGridSpacing / 5; // 5 minor squares per major square

    const leadColor = '#1a56db'; // Blue for the ECG line

    const centerY = canvas.height / 2; // Vertical center of the canvas for baseline

    // Draw minor grid lines

    ctx.strokeStyle = minorGridColor;

    ctx.lineWidth = 0.2;

    for (let x = 0; x < canvas.width; x += minorGridSpacing) {

      ctx.beginPath();

      ctx.moveTo(x, 0);

      ctx.lineTo(x, canvas.height);

      ctx.stroke();

    }

    for (let y = 0; y < canvas.height; y += minorGridSpacing) {

      ctx.beginPath();

      ctx.moveTo(0, y);

      ctx.lineTo(canvas.width, y);

      ctx.stroke();

    }

    // Draw major grid lines

    ctx.strokeStyle = majorGridColor;

    ctx.lineWidth = 0.5;

    for (let x = 0; x < canvas.width; x += majorGridSpacing) {

      ctx.beginPath();

      ctx.moveTo(x, 0);

      ctx.lineTo(x, canvas.height);

      ctx.stroke();

    }

    for (let y = 0; y < canvas.height; y += majorGridSpacing) {

      ctx.beginPath();

      ctx.moveTo(0, y);

      ctx.lineTo(canvas.width, y);

      ctx.stroke();

    }

    // Draw the ECG waveform

    if (ecgData.length > 1) {

      ctx.strokeStyle = leadColor;

      ctx.lineWidth = 2; // Thicker line for the ECG trace

      ctx.beginPath();

      // Start the path at the first point

      ctx.moveTo(0, centerY - ecgData[0]);

      // Draw lines to connect subsequent points

      // The x-coordinate is simply the index, scaled to fit the canvas width

      // The y-coordinate is the baseline (centerY) minus the ECG data value (to invert y-axis for typical ECG display)

      for (let i = 1; i < ecgData.length; i++) {

        const x = i;

        const y = centerY - ecgData[i];

        ctx.lineTo(x, y);

      }

      ctx.stroke(); // Render the ECG path

    }

  }, [ecgData]); // Redraw whenever ecgData changes

  return (

    <div className="w-full flex justify-center mt-6">

      <canvas

        ref={canvasRef}

        className="border border-gray-300 rounded-md bg-white shadow-inner w-full h-64" // Responsive sizing

      ></canvas>

    </div>

  );

}

export default App;