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

import { FaStar } from "react-icons/fa";

import { createClient } from '@supabase/supabase-js';

import { useRef } from 'react';

// Initialisation de Supabase

const supabase = createClient(

  import.meta.env.VITE\_SUPABASE\_URL,

  import.meta.env.VITE\_SUPABASE\_ANON\_KEY

);

const ModernMoodTrackerApp = () => {

  // États existants

  const [entries, setEntries] = useState([]);

  const [note, setNote] = useState('');

  const [mood, setMood] = useState(null);

  const [sleep, setSleep] = useState(0);

  const [stress, setStress] = useState(0);

  const [activity, setActivity] = useState(0);

  const [energy, setEnergy] = useState(0);

  const [spirituality, setSpirituality] = useState(0);

  const [cgm, setCGM] = useState(0);

  const [definition, setDefinition] = useState('اختر عاطفة لترى تعريفها');

  const [emotionDefinitions, setEmotionDefinitions] = useState({});

  const [successMessage, setSuccessMessage] = useState("");

  const [isErrorMessage, setIsErrorMessage] = useState(false);

  const [isOnline, setIsOnline] = useState(navigator.onLine);

  // Composant étoiles modernisé

  const ModernStarRating = ({ value, onChange }) => (

    <div className="flex gap-1">

      {[...Array(5)].map((\_, index) => (

        <FaStar

          key={index}

          size={20}

          className="cursor-pointer transition-colors duration-200"

          style={{ color: index + 1 <= value ? "#3B82F6" : "#E5E7EB" }}

          onClick={() => onChange(index + 1)}

        />

      ))}

    </div>

  );

  // Gestion online/offline

  useEffect(() => {

    const handleOnline = () => {

      console.log("✅ Connexion rétablie - Tentative de synchronisation...");

      setIsOnline(true);

      syncLocalEntries();

    };

    const handleOffline = () => {

      console.log("❌ Connexion perdue");

      setIsOnline(false);

    };

    window.addEventListener("online", handleOnline);

    window.addEventListener("offline", handleOffline);

    return () => {

      window.removeEventListener("online", handleOnline);

      window.removeEventListener("offline", handleOffline);

    };

  }, []);

  // Chargement des définitions des émotions

  useEffect(() => {

    const loadEmotionDefinitions = async () => {

      console.log('Chargement des définitions...');

      if (isOnline) {

        try {

          console.log('Tentative de récupération depuis Supabase...');

          const { data, error } = await supabase

            .from('emotions')

            .select('\*');

          if (error) {

            console.error('Erreur Supabase:', error);

            throw error;

          }

          console.log('Données reçues de Supabase:', data);

          // Création d'un objet avec les émotions comme clés

          const definitions = {};

          data.forEach(item => {

            // Utilisez le nom de la colonne exacte de votre table Supabase

            if (item.emotion\_fr && item.definition\_ar) {

              definitions[item.emotion\_fr] = item.definition\_ar;

            }

          });

          console.log('Définitions formatées:', definitions);

          setEmotionDefinitions(definitions);

          localStorage.setItem('emotionDefinitions', JSON.stringify(definitions));

        } catch (error) {

          console.error('Erreur chargement définitions:', error);

          const localDefinitions = JSON.parse(localStorage.getItem('emotionDefinitions')) || {};

          setEmotionDefinitions(localDefinitions);

        }

      } else {

        console.log('Mode hors ligne, chargement depuis le stockage local...');

        const localDefinitions = JSON.parse(localStorage.getItem('emotionDefinitions')) || {};

        setEmotionDefinitions(localDefinitions);

      }

    };

    loadEmotionDefinitions();

  }, [isOnline]);

  // Gestion du statut online/offline et synchronisation

  useEffect(() => {

    const loadDefinitions = async () => {

      if (isOnline) {

        try {

          console.log("Chargement initial des définitions");

          const { data, error } = await supabase

            .from('emotions')

            .select('emotion\_fr, definition');

          if (error) throw error;

          const definitions = {};

          data.forEach(item => {

            definitions[item.emotion\_fr] = item.definition;

          });

          console.log("Définitions chargées:", definitions);

          localStorage.setItem("emotionDefinitions", JSON.stringify(definitions));

        } catch (error) {

          console.error("Erreur de chargement des définitions:", error);

        }

      }

    };

    loadDefinitions();

  }, [isOnline]);

  // Chargement initial des données

  useEffect(() => {

    const loadData = async () => {

      if (isOnline) {

        try {

          const { data, error } = await supabase

            .from('mood\_entries')

            .select('\*')

            .order('date', { ascending: false });

          if (error) throw error;

          setEntries(data || []);

        } catch (error) {

          console.error('Erreur de chargement:', error);

          // Charger les données locales en cas d'erreur

          const localData = JSON.parse(localStorage.getItem('moodEntries')) || [];

          setEntries(localData);

        }

      } else {

        const localData = JSON.parse(localStorage.getItem('moodEntries')) || [];

        setEntries(localData);

      }

    };

    loadData();

  }, [isOnline]);

  useEffect(() => {

    console.log("💾 Vérification des entrées locales au chargement...");

    console.log("📂 Données locales disponibles :", JSON.parse(localStorage.getItem("moodEntries")) || []);

  }, []);

  const isSyncingRef = useRef(false);

    const syncLocalEntries = async () => {

      if (isSyncingRef.current) {

        console.log("⚠️ Synchronisation déjà en cours, annulation.");

        return;

      }

      if (!navigator.onLine) {

        console.log("❌ Hors ligne - Synchronisation impossible.");

        return;

      }

      let localEntries = JSON.parse(localStorage.getItem("moodEntries")) || [];

      console.log("📂 Entrées locales trouvées :", localEntries);

      if (localEntries.length === 0) {

        console.log("✅ Aucune entrée locale à synchroniser.");

        return;

      }

      isSyncingRef.current = true; // 🔒 Verrouille la synchronisation

      console.log(`📤 Envoi de ${localEntries.length} entrées à Supabase...`);

      try {

        const { error } = await supabase

          .from('mood\_entries')

          .insert(localEntries);

        if (error) {

          throw error;

        }

        console.log("✅ Synchronisation réussie !");

        localStorage.removeItem("moodEntries"); // ✅ Nettoyage après envoi

        setEntries((prev) => [...localEntries, ...prev]);

        setSuccessMessage("📡 Données synchronisées avec succès !");

      } catch (error) {

        console.error("❌ Erreur lors de la synchronisation avec Supabase :", error);

        setSuccessMessage("❌ Erreur lors de la synchronisation !");

      }

      isSyncingRef.current = false; // 🔓 Déverrouille la synchronisation

      setTimeout(() => setSuccessMessage(""), 3000);

    };

// 🔄 Écouteur d'événement pour synchroniser dès que l'utilisateur repasse en ligne

useEffect(() => {

  window.addEventListener("online", syncLocalEntries);

  return () => {

    window.removeEventListener("online", syncLocalEntries);

  };

}, []);

  // Sauvegarde locale

  const saveLocally = (entry) => {

    const localEntries = JSON.parse(localStorage.getItem('moodEntries')) || [];

    localEntries.unshift(entry);

    localStorage.setItem('moodEntries', JSON.stringify(localEntries));

    setEntries(prev => [entry, ...prev]);

    setSuccessMessage('Entrée sauvegardée localement');

  };

  // Réinitialisation des champs

  const resetFields = () => {

    setMood(null);

    setNote('');

    setSleep(0);

    setStress(0);

    setActivity(0);

    setEnergy(0);

    setSpirituality(0);

    setCGM(0);

    setDefinition("");  // ✅ Efface la zone de texte de la définition

    setTimeout(() => setSuccessMessage(''), 3000);

  };

  // Configuration des catégories

  const categories = [

    { icon: "/icons/spirituality.png", label: <span style={{ fontFamily: "'Amiri'" }}>ورد القرآن</span>, state: spirituality, setter: setSpirituality },

    { icon: "/icons/cgm.png", label: <span style={{ fontFamily: "'Amiri'" }}>ورد الأذكار</span>, state: cgm, setter: setCGM },

    { icon: "/icons/sleep.png", label: <span style={{ fontFamily: "'Amiri'" }}> السنن الرواتب </span>, state: sleep, setter: setSleep },

    { icon: "/icons/stress.png", label: <span style={{ fontFamily: "'Amiri'" }}> الباقيات الصالحات </span>, state: stress, setter: setStress },

    { icon: "/icons/activity.png", label: <span style={{ fontFamily: "'Amiri'" }}> قيام الليل </span>, state: activity, setter: setActivity },

    { icon: "/icons/energy.png", label: <span style={{ fontFamily: "'Amiri'" }}> أعمال مأجورة </span>, state: energy, setter: setEnergy }

  ];

  // Configuration des émotions

  const emotions = [

    { name: 'الملل', file: 'Mood\_Ennui.png' },

    { name: 'المفاجأة', file: 'Mood\_Surprise.png' },

    { name: 'الرضا', file: 'Mood\_Satisfaction.png' },

    { name: 'الحماس', file: 'Mood\_Excitation.png' },

    { name: 'السعادة', file: 'Mood\_Bonheur.png' },

    { name: 'الغضب', file: 'Mood\_Colere.png' },

    { name: 'الحزن', file: 'Mood\_Tristesse.png' },

    { name: 'الاشمئزاز', file: 'Mood\_Degout.png' },

    { name: 'الخوف', file: 'Mood\_Peur.png' },

    { name: 'القلق', file: 'Mood\_Anxiete.png' }

  ];

  // Fonction de sauvegarde d'une entrée

  const handleSave = async () => {

    if (!note.trim() || !mood) {

      setIsErrorMessage(true);

      setSuccessMessage("الرجاء إدخال ملاحظة واختيار العاطفة");

      setTimeout(() => {

        setSuccessMessage("");

        setIsErrorMessage(false);

      }, 3000);

      return;

    }

    const newEntry = {

      mood,

      note,

      sleep,

      stress,

      activity,

      energy,

      spirituality,

      cgm,

      date: new Date().toISOString().split('T')[0],

      heure: new Date().toLocaleTimeString('fr-FR', {

        hour: '2-digit',

        minute: '2-digit',

        second: '2-digit'

      })

    };

    // Récupération des entrées existantes

    let localEntries = JSON.parse(localStorage.getItem("moodEntries")) || [];

    if (!isOnline) {

      console.log("Mode hors ligne - Sauvegarde locale");

      // Ajout de la nouvelle entrée au début du tableau

      localEntries.unshift(newEntry);

      // Sauvegarde dans localStorage

      localStorage.setItem("moodEntries", JSON.stringify(localEntries));

      setIsErrorMessage(false);

      setSuccessMessage("Entrée sauvegardée localement");

      // Mise à jour de l'état

      setEntries([newEntry, ...entries]);

    } else {

      try {

        console.log("Mode en ligne - Sauvegarde dans Supabase");

        const { error } = await supabase

          .from('mood\_entries')

          .insert([newEntry]);

        if (error) throw error;

        setIsErrorMessage(false);

        setSuccessMessage("لقد تم التسجيل بنجاح  !");

        // Mise à jour de l'état

        setEntries([newEntry, ...entries]);

      } catch (error) {

        console.error("Erreur de sauvegarde Supabase:", error);

        // Fallback vers stockage local en cas d'erreur

        localEntries.unshift(newEntry);

        localStorage.setItem("moodEntries", JSON.stringify(localEntries));

        setIsErrorMessage(false);

        setSuccessMessage("Erreur - Entrée sauvegardée localement");

        // Mise à jour de l'état

        setEntries([newEntry, ...entries]);

      }

    }

    // Réinitialisation des champs

    setMood(null);

    setNote("");

    setSleep(0);

    setStress(0);

    setActivity(0);

    setEnergy(0);

    setSpirituality(0);

    setCGM(0);

    setDefinition("");  // ✅ Efface la zone de texte de la définition

    setIsErrorMessage(false);

    setTimeout(() => setSuccessMessage(""), 3000);

  };

  const handleEmotionClick = async (emotionName) => {

    console.log("Émotion sélectionnée:", emotionName);

    setMood(emotionName);

    if (!isOnline) {

      console.log("Mode hors ligne - Recherche dans le localStorage");

      const savedDefinitions = JSON.parse(localStorage.getItem("emotionDefinitions")) || {};

      const savedDefinition = savedDefinitions[emotionName];

      if (savedDefinition) {

        console.log("Définition trouvée dans le localStorage:", savedDefinition);

        setDefinition(savedDefinition);

      } else {

        setDefinition("Définition non disponible en mode hors ligne");

      }

      return;

    }

    try {

      const { data, error } = await supabase

        .from('emotions')

        .select('definition')

        .eq('emotion\_fr', emotionName)

        .single();

      if (error) {

        console.error("Erreur Supabase:", error);

        throw error;

      }

      console.log("Données reçues de Supabase:", data);

      if (data) {

        // Sauvegarder la définition dans le localStorage

        const savedDefinitions = JSON.parse(localStorage.getItem("emotionDefinitions")) || {};

        savedDefinitions[emotionName] = data.definition;

        localStorage.setItem("emotionDefinitions", JSON.stringify(savedDefinitions));

        setDefinition(data.definition);

      } else {

        setDefinition('Définition non disponible');

      }

    } catch (error) {

      console.error("Erreur:", error);

      setDefinition('Erreur lors de la récupération de la définition');

    }

  };

  return (

    <div className="min-h-screen bg-gray-50 p-4 md:p-6">

      <div className="max-w-4xl mx-auto bg-white rounded-lg shadow-lg p-6">

        {/\* En-tête avec indicateur de statut \*/}

        <div className="text-center space-y-2 mb-6">

          <div className="flex items-center justify-center gap-2">

            <h1 className="text-2xl font-bold bg-gradient-to-r from-blue-600 to-blue-400 inline-block text-transparent bg-clip-text">

              SpiritualitySky

            </h1>

            <span className={`h-2 w-2 rounded-full ${isOnline ? 'bg-green-500' : 'bg-red-500'}`} />

          </div>

          <p className="text-xl font-arabic text-gray-800">يوميات مناخك الروحي</p>

          <p className="text-sm text-gray-500">

            {new Date().toLocaleString('fr-FR', {

              hour: '2-digit',

              minute: '2-digit',

              second: '2-digit',

              hour12: true

            })}

          </p>

        </div>

        {successMessage && (

          <div className={`mb-4 p-3 rounded-md ${isErrorMessage ? 'bg-red-100 text-red-700' : 'bg-green-100 text-green-700'}`}>

            {successMessage}

          </div>

        )}

        <div className="space-y-6">

          {/\* Grille des catégories \*/}

          <div className="grid grid-cols-2 md:grid-cols-3 gap-4">

            {categories.map((category, index) => (

              <div key={index} className="bg-gray-50 rounded-lg p-3 transition-all hover:bg-gray-100">

                <div className="flex items-center gap-2 mb-2">

                  <img

                    src={category.icon}

                    alt={category.label}

                    className="w-5 h-5 object-contain"

                  />

                  <span style={{ fontFamily: "'Amiri'" }}>{category.label}</span>

                </div>

                <ModernStarRating value={category.state} onChange={category.setter} />

              </div>

            ))}

          </div>

          {/\* Section des émotions \*/}

          <div className="bg-gray-50 p-4 rounded-lg">

            <div className="grid grid-cols-5 gap-4">

              {emotions.map((emotion) => (

                <div

                  key={emotion.name}

                  className="flex flex-col items-center group cursor-pointer"

                  onClick={() => handleEmotionClick(emotion.name)}

                >

                  <div className={`

                    relative p-2 rounded-full transition-all duration-200

                    ${mood === emotion.name ? 'bg-blue-100 scale-110' : 'hover:bg-gray-100 hover:scale-105'}

                  `}>

                    <img

                      src={`/icons/${emotion.file}`}

                      alt={emotion.name}

                      className="w-8 h-8 object-contain"

                    />

                  </div>

                  <span style={{ fontFamily: "'Amiri'" }}>{emotion.name}</span>

                </div>

              ))}

            </div>

                          {definition && (

              <p className="mt-4 text-l font-arabic text-gray-600 bg-white p-3 rounded-md border border-gray-200 text-right">

                {definition}

              </p>

            )}

          </div>

          {/\* Zone de commentaire \*/}

          <textarea

            placeholder=" كيف كان يومك ؟ وماهي أهم إنجازاتك اليوم ؟"

            value={note}

            onChange={(e) => setNote(e.target.value)}

            className="w-full min-h-[100px] p-3 text-right border border-gray-200 rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-transparent resize-none font-[Amiri]"

          />

          {/\* Boutons d'action \*/}

          <div className="flex flex-col gap-2">

            <button

              onClick={handleSave}

              className="w-full bg-blue-600 hover:bg-blue-700 text-white font-arabic py-2 px-6 rounded-lg transition-colors duration-200"

            >

              التسجيل

            </button>

            {/\* ✅ Bouton de synchronisation \*/}

            <button

              onClick={() => {

                localStorage.removeItem("moodEntries");

                setEntries([]);

              }}

              className="w-full border border-red-600 text-red-600 font-arabic hover:bg-red-50 py-2 px-4 rounded-lg transition-colors duration-200"

            >

              حذف البيانات

            </button>

          </div>

        </div>

      </div>

    </div>

  );

};

export default ModernMoodTrackerApp;