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

import { Guardian, Child, ModalState, View, StaffVerificationState, AppDocument, EventLog } from './types';

import { auth, db, storage, messaging } from './firebase';

import { RecaptchaVerifier, signInWithPhoneNumber, ConfirmationResult, signOut } from 'firebase/auth';

import { collection, doc, getDocs, onSnapshot, query, where, updateDoc, addDoc, deleteDoc, serverTimestamp } from 'firebase/firestore';

import { ref, uploadBytes, getDownloadURL, deleteObject } from 'firebase/storage';

import { getToken } from 'firebase/messaging';

import LoginPage from './components/LoginPage';

import ChildListPage from './components/ChildListPage';

import StaffVerifyPage from './components/StaffVerifyPage';

import AdminPage from './components/AdminPage';

import Modal from './components/Modal';

import { CheckIcon } from './components/icons/CheckIcon';

import StaffDashboardPage from './components/StaffDashboardPage';

import AdminDashboardPage from './components/AdminDashboardPage';

import OtpPage from './components/OtpPage';

import CommunicationsPage from './components/CommunicationsPage';

import StaffLoginPage from './components/StaffLoginPage';

declare global {

  interface Window {

    recaptchaVerifier?: RecaptchaVerifier;

    confirmationResult?: ConfirmationResult;

  }

}

declare var grecaptcha: {

  reset: (widgetId?: number) => void;

} | undefined;

// Add role to Guardian type

type User = Guardian;

// Context for shared state

interface AppContextType {

  currentUser: User | null;

  children: Child[];

  allGuardians: Record<string, Guardian>;

  documents: AppDocument[];

  eventLogs: EventLog[];

  error: string;

  setError: React.Dispatch<React.SetStateAction<string>>;

  displaySuccessMessage: (message: string) => void;

  handleLogout: () => void;

  handleCheckIn: (childId: string) => Promise<void>;

  handleShowCode: (childId: string) => void;

  handleVerifyCode: (code: string) => Promise<void>;

  handleStaffCheckout: (childId: string) => Promise<void>;

  handleAdminToggleStatus: (childId: string) => Promise<void>;

  handleUploadDocument: (file: File, name: string) => Promise<void>;

  handleDeleteDocument: (documentToDelete: AppDocument) => Promise<void>;

  handleSendNotification: (title: string, body: string) => Promise<void>;

  handleEnableNotifications: () => Promise<void>;

  exportChildrenData: () => void;

  exportEventLogs: () => void;

}

const AppContext = createContext<AppContextType | undefined>(undefined);

export const useAppContext = () => {

  const context = useContext(AppContext);

  if (!context) throw new Error('useAppContext must be used within AppProvider');

  return context;

};

const generateSecure6DigitCode = (): string => {

  if (window.crypto && window.crypto.getRandomValues) {

    const array = new Uint32Array(1);

    window.crypto.getRandomValues(array);

    const max = Math.pow(2, 32);

    const range = 900000;

    const randomInRange = Math.floor((array[0] / max) \* range) + 100000;

    return randomInRange.toString().padStart(6, '0');

  }

  // Fallback for older browsers

  return Math.floor(100000 + Math.random() \* 900000).toString();

};

const App: React.FC = () => {

  const [view, setView] = useState<View>('login');

  const [currentUser, setCurrentUser] = useState<User | null>(null);

  const [children, setChildren] = useState<Child[]>([]);

  const [allGuardians, setAllGuardians] = useState<Record<string, Guardian>>({});

  const [documents, setDocuments] = useState<AppDocument[]>([]);

  const [eventLogs, setEventLogs] = useState<EventLog[]>([]);

  const [error, setError] = useState<string>('');

  const [modalState, setModalState] = useState<ModalState>({ isOpen: false });

  const [showSuccess, setShowSuccess] = useState(false);

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

  const [staffVerificationState, setStaffVerificationState] = useState<StaffVerificationState>({ status: 'idle' });

  const [phoneToVerify, setPhoneToVerify] = useState<string | null>(null);

  const [confirmationResult, setConfirmationResult] = useState<ConfirmationResult | null>(null);

  // Real-time data listeners

  useEffect(() => {

    const childrenQuery = query(collection(db, 'children'));

    const unsubscribeChildren = onSnapshot(childrenQuery, snapshot => {

      const childrenData = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() } as Child));

      setChildren(childrenData);

    });

    const docsQuery = query(collection(db, 'documents'));

    const unsubscribeDocs = onSnapshot(docsQuery, snapshot => {

      const docsData = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() } as AppDocument));

      setDocuments(docsData);

    });

    const eventsQuery = query(collection(db, 'events'));

    const unsubscribeEvents = onSnapshot(eventsQuery, snapshot => {

      const eventsData = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() } as EventLog));

      setEventLogs(eventsData);

    });

    return () => {

      unsubscribeChildren();

      unsubscribeDocs();

      unsubscribeEvents();

    };

  }, []);

  // Fetch all guardians when admin dashboard is opened

  useEffect(() => {

    if (view === 'admin-dashboard' && Object.keys(allGuardians).length === 0) {

      const fetchGuardians = async () => {

        try {

          const querySnapshot = await getDocs(collection(db, 'guardians'));

          const guardiansData: Record<string, Guardian> = {};

          querySnapshot.forEach((doc) => {

            const data = doc.data();

            guardiansData[doc.id] = { id: doc.id, ...data } as Guardian;

          });

          setAllGuardians(guardiansData);

        } catch (err) {

          setError('Failed to load guardian data.');

        }

      };

      fetchGuardians();

    }

  }, [view, allGuardians]);

  const setupRecaptcha = () => {

    if (!window.recaptchaVerifier) {

      window.recaptchaVerifier = new RecaptchaVerifier(auth, 'recaptcha-container', {

        'size': 'invisible',

        'callback': () => { /\* reCAPTCHA solved \*/ }

      });

    }

  };

  const handleLoginRequest = async (phone: string) => {

    try {

      setError('');

      setupRecaptcha();

      const formattedPhone = `+1${phone}`;

      const confirmation = await signInWithPhoneNumber(auth, formattedPhone, window.recaptchaVerifier!);

      setConfirmationResult(confirmation);

      setPhoneToVerify(phone);

      setView('login-otp');

    } catch (err: any) {

      console.error('Firebase Auth Error:', err);

      setError('Failed to send verification code. Please check the phone number and try again.');

      if (typeof grecaptcha !== 'undefined') {

        window.recaptchaVerifier?.render().then(widgetId => grecaptcha?.reset(widgetId));

      }

    }

  };

  const handleVerifyOtp = async (otp: string) => {

    if (!confirmationResult) {

      setError('Verification process timed out. Please try again.');

      return;

    }

    try {

      const result = await confirmationResult.confirm(otp);

      const userPhone = result.user.phoneNumber;

      if (userPhone) {

        const plainPhone = userPhone.slice(2); // Remove +1

        const q = query(collection(db, 'guardians'), where('phone', '==', plainPhone));

        const querySnapshot = await getDocs(q);

        if (!querySnapshot.empty) {

          const userDoc = querySnapshot.docs[0];

          const userData = { id: userDoc.id, ...userDoc.data() } as User;

          setCurrentUser(userData);

          if (userData.role === 'staff' || userData.role === 'admin') {

            setView('staff-dashboard');

          } else {

            setView('child-list');

          }

          setError('');

          setPhoneToVerify(null);

        } else {

          setError('User profile not found for this phone number.');

          handleLogout();

        }

      }

    } catch (err: any) {

      console.error('OTP Verification Error:', err);

      setError('Invalid code or an error occurred. Please try again.');

    }

  };

  const handleLogout = async () => {

    try {

      await signOut(auth);

    } catch (err) {

      console.error('Logout Error:', err);

    }

    setCurrentUser(null);

    setStaffVerificationState({ status: 'idle' });

    setView('login');

    setPhoneToVerify(null);

  };

  const handleBackToLogin = () => {

    setView('login');

    setError('');

    setPhoneToVerify(null);

  };

  const displaySuccessMessage = (message: string) => {

    setSuccessMessage(message);

    setShowSuccess(true);

    setTimeout(() => {

      setShowSuccess(false);

      setSuccessMessage('');

    }, 3000);

  };

  const logEvent = async (type: 'check-in' | 'check-out', childId: string) => {

    if (!currentUser) return;

    try {

      await addDoc(collection(db, 'events'), {

        type,

        childId,

        byUserId: currentUser.id,

        byUserName: currentUser.name,

        byRole: currentUser.role,

        timestamp: serverTimestamp(),

      });

    } catch (err) {

      console.error('Event Logging Error:', err);

      setError('Failed to log event.');

    }

  };

  const handleCheckIn = async (childId: string) => {

    try {

      const newCode = generateSecure6DigitCode();

      const childRef = doc(db, 'children', childId);

      await updateDoc(childRef, {

        status: 'in',

        pickupCode: newCode,

        lastActionTimestamp: serverTimestamp(),

      });

      await logEvent('check-in', childId);

      setModalState({ isOpen: true, mode: 'show-code', childId, code: newCode });

    } catch (err) {

      console.error('Check-In Error:', err);

      setError('Failed to check in child.');

    }

  };

  const handleShowCode = (childId: string) => {

    const child = children.find(c => c.id === childId);

    if (child?.status === 'in' && child.pickupCode) {

      setModalState({ isOpen: true, mode: 'show-code', childId, code: child.pickupCode });

    }

  };

  const handleVerifyCode = async (code: string) => {

    setStaffVerificationState({ status: 'loading' });

    try {

      const foundChild = children.find(c => c.status === 'in' && c.pickupCode === code);

      if (foundChild) {

        const q = query(collection(db, 'guardians'), where('childIds', 'array-contains', foundChild.id));

        const querySnapshot = await getDocs(q);

        const foundGuardian = querySnapshot.empty ? undefined : { id: querySnapshot.docs[0].id, ...querySnapshot.docs[0].data() } as Guardian;

        setStaffVerificationState({ status: 'found', child: foundChild, guardian: foundGuardian });

      } else {

        setStaffVerificationState({ status: 'not-found', message: `No active check-in found for code "${code}".` });

      }

    } catch (err) {

      console.error('Verify Code Error:', err);

      setStaffVerificationState({ status: 'error', message: 'Error verifying code.' });

    }

  };

  const handleStaffCheckout = async (childId: string) => {

    try {

      const childRef = doc(db, 'children', childId);

      await updateDoc(childRef, {

        status: 'out',

        pickupCode: null,

        lastActionTimestamp: serverTimestamp(),

      });

      await logEvent('check-out', childId);

      const child = children.find(c => c.id === childId);

      setStaffVerificationState({ status: 'checked-out', message: `${child?.name || 'Child'} has been successfully checked out.` });

    } catch (err) {

      console.error('Checkout Error:', err);

      setError('Failed to check out child.');

    }

  };

  const handleAdminToggleStatus = async (childId: string) => {

    const child = children.find(c => c.id === childId);

    if (!child) return;

    const childRef = doc(db, 'children', childId);

    try {

      if (child.status === 'out') {

        const newCode = generateSecure6DigitCode();

        await updateDoc(childRef, {

          status: 'in',

          pickupCode: newCode,

          lastActionTimestamp: serverTimestamp(),

        });

        await logEvent('check-in', childId);

        displaySuccessMessage(`${child.name} has been manually checked in.`);

      } else {

        await updateDoc(childRef, {

          status: 'out',

          pickupCode: null,

          lastActionTimestamp: serverTimestamp(),

        });

        await logEvent('check-out', childId);

        displaySuccessMessage(`${child.name} has been manually checked out.`);

      }

    } catch (err) {

      console.error('Toggle Status Error:', err);

      setError('Failed to toggle child status.');

    }

  };

  const handleUploadDocument = async (file: File, name: string) => {

    const docName = name || file.name;

    const storageRef = ref(storage, `documents/${Date.now()}\_${docName}`);

    try {

      await uploadBytes(storageRef, file);

      const url = await getDownloadURL(storageRef);

      await addDoc(collection(db, 'documents'), {

        name: docName,

        url: url,

        storagePath: storageRef.fullPath,

        uploadedAt: serverTimestamp(),

      });

      displaySuccessMessage('Document uploaded successfully!');

    } catch (err) {

      console.error('Upload Document Error:', err);

      setError('Failed to upload document.');

    }

  };

  const handleDeleteDocument = async (documentToDelete: AppDocument) => {

    try {

      await deleteDoc(doc(db, 'documents', documentToDelete.id));

      if (documentToDelete.storagePath) {

        const storageRef = ref(storage, documentToDelete.storagePath);

        await deleteObject(storageRef);

      }

      displaySuccessMessage('Document deleted.');

    } catch (err) {

      console.error('Delete Document Error:', err);

      setError('Failed to delete document.');

    }

  };

  const handleSendNotification = async (title: string, body: string) => {

    try {

      await addDoc(collection(db, 'notifications'), {

        title,

        body,

        createdAt: serverTimestamp()

      });

      displaySuccessMessage('Notification sent!');

    } catch (err) {

      console.error('Send Notification Error:', err);

      setError('Failed to send notification.');

    }

  };

  const handleEnableNotifications = async () => {

    if (!('Notification' in window) || !messaging || !currentUser) return;

    try {

      const permission = await Notification.requestPermission();

      if (permission === 'granted') {

        const vapidKey = "YOUR\_VAPID\_KEY\_FROM\_FIREBASE\_SETTINGS"; // IMPORTANT: Replace with your actual VAPID key

        if (!vapidKey || vapidKey === "YOUR\_VAPID\_KEY\_FROM\_FIREBASE\_SETTINGS") {

          setError('VAPID key not configured. Notifications will not be sent.');

          return;

        }

        const fcmToken = await getToken(messaging, { vapidKey });

        if (fcmToken) {

          const userRef = doc(db, 'guardians', currentUser.id);

          const currentTokens = currentUser.fcmTokens || [];

          if (!currentTokens.includes(fcmToken)) {

            await updateDoc(userRef, {

              fcmTokens: [...currentTokens, fcmToken]

            });

          }

          displaySuccessMessage('Notifications enabled!');

        }

      } else {

        setError('Notification permission was denied.');

      }

    } catch (err) {

      console.error('Enable Notifications Error:', err);

      setError('Could not enable notifications.');

    }

  };

  const exportToCSV = (data: any[], filename: string) => {

      const replacer = (key: any, value: any) => value === null || value === undefined ? '' : value;

      const header = Object.keys(data[0]);

      const csv = [

        header.join(','),

        ...data.map(row => header.map(fieldName => JSON.stringify(row[fieldName], replacer)).join(','))

      ].join('\r\n');

      const blob = new Blob([csv], { type: 'text/csv;charset=utf-8;' });

      const link = document.createElement('a');

      if (link.download !== undefined) {

        const url = URL.createObjectURL(blob);

        link.setAttribute('href', url);

        link.setAttribute('download', filename);

        link.style.visibility = 'hidden';

        document.body.appendChild(link);

        link.click();

        document.body.removeChild(link);

      }

  };

  const exportChildrenData = () => {

    const data = children.map(child => ({

      id: child.id,

      name: child.name,

      troop: child.troop,

      rank: child.rank,

      status: child.status,

      lastActionTimestamp: child.lastActionTimestamp?.toDate().toISOString() || 'N/A'

    }));

    if (data.length > 0) {

      exportToCSV(data, 'children\_data.csv');

    } else {

      displaySuccessMessage("No children data to export.");

    }

  };

  const exportEventLogs = () => {

    const data = eventLogs.map(log => ({

      id: log.id,

      type: log.type,

      childId: log.childId,

      byUserId: log.byUserId,

      byUserName: log.byUserName,

      byRole: log.byRole,

      timestamp: log.timestamp?.toDate().toISOString() || 'N/A'

    }));

     if (data.length > 0) {

      exportToCSV(data, 'event\_logs.csv');

    } else {

      displaySuccessMessage("No event logs to export.");

    }

  };

  const resetStaffVerification = () => setStaffVerificationState({ status: 'idle' });

  const closeModal = () => setModalState({ isOpen: false });

  const userChildren = currentUser ? children.filter(c => currentUser.childIds?.includes(c.id)) : [];

  const contextValue: AppContextType = {

    currentUser,

    children,

    allGuardians,

    documents,

    eventLogs,

    error,

    setError,

    displaySuccessMessage,

    handleLogout,

    handleCheckIn,

    handleShowCode,

    handleVerifyCode,

    handleStaffCheckout,

    handleAdminToggleStatus,

    handleUploadDocument,

    handleDeleteDocument,

    handleSendNotification,

    handleEnableNotifications,

    exportChildrenData,

    exportEventLogs,

  };

  const renderContent = () => {

    // Staff/Admin views require a specific role

    const canAccessStaffTools = currentUser?.role === 'staff' || currentUser?.role === 'admin';

    const canAccessAdminTools = currentUser?.role === 'admin';

    switch (view) {

      case 'login':

        return <LoginPage onLogin={handleLoginRequest} error={error} setError={setError} onSwitchView={setView} />;

      case 'login-otp':

        return <OtpPage phone={phoneToVerify} onVerify={handleVerifyOtp} error={error} setError={setError} onBack={handleBackToLogin} />;

      case 'child-list':

        return currentUser && <ChildListPage guardian={currentUser} children={userChildren} documents={documents} onCheckIn={handleCheckIn} onShowCode={handleShowCode} onLogout={handleLogout} onEnableNotifications={handleEnableNotifications} />;

      case 'staff-login':

          return <StaffLoginPage onLogin={handleLoginRequest} error={error} setError={setError} onBack={handleBackToLogin} />;

      case 'staff-dashboard':

        return canAccessStaffTools ? <StaffDashboardPage onSwitchView={setView} onLogout={handleLogout} isAdmin={canAccessAdminTools} /> : <LoginPage onLogin={handleLoginRequest} error="Access Denied" setError={setError} onSwitchView={setView} />;

      case 'staff-verify':

        return canAccessStaffTools ? <StaffVerifyPage verificationState={staffVerificationState} onVerify={handleVerifyCode} onCheckout={handleStaffCheckout} onReset={resetStaffVerification} onBack={() => setView('staff-dashboard')} /> : <LoginPage onLogin={handleLoginRequest} error="Access Denied" setError={setError} onSwitchView={setView} />;

      case 'admin-dashboard':

        return canAccessAdminTools ? <AdminDashboardPage allChildren={children} allGuardians={allGuardians} onToggleStatus={handleAdminToggleStatus} onBack={() => setView('staff-dashboard')} exportChildrenData={exportChildrenData} exportEventLogs={exportEventLogs} /> : <LoginPage onLogin={handleLoginRequest} error="Access Denied" setError={setError} onSwitchView={setView} />;

      case 'communications':

        return canAccessAdminTools ? <CommunicationsPage documents={documents} onUpload={handleUploadDocument} onDelete={handleDeleteDocument} onSendNotification={handleSendNotification} onBack={() => setView('staff-dashboard')} /> : <LoginPage onLogin={handleLoginRequest} error="Access Denied" setError={setError} onSwitchView={setView} />;

      case 'admin':

        return <AdminPage onBack={handleBackToLogin} />;

      default:

        return <LoginPage onLogin={handleLoginRequest} error={error} setError={setError} onSwitchView={setView} />;

    }

  };

  return (

    <AppContext.Provider value={contextValue}>

      <div className="bg-gray-50 min-h-screen text-gray-800">

        <div id="recaptcha-container"></div>

        <div className={`fixed top-5 right-5 bg-green-500 text-white py-3 px-5 rounded-lg shadow-xl flex items-center transition-transform duration-300 z-50 ${showSuccess ? 'translate-x-0 opacity-100' : 'translate-x-full opacity-0'}`}>

          <CheckIcon className="w-6 h-6 mr-2" />

          <span className="font-medium">{successMessage}</span>

        </div>

        <div className="container mx-auto max-w-lg p-4">

          {renderContent()}

        </div>

        {modalState.isOpen && modalState.mode === 'show-code' && (

          <Modal onClose={closeModal}>

            <div className="p-2 text-center">

              <h2 className="text-2xl font-bold mb-4 text-green-800">Your Pick-Up Code</h2>

              <p className="text-gray-600 mb-2">Present this one-time code to a staff member for pick-up:</p>

              <p className="text-5xl font-bold tracking-widest bg-gray-100 text-green-700 py-4 px-6 rounded-lg my-4 inline-block">

                {modalState.code}

              </p>

              <button

                onClick={closeModal}

                className="mt-6 w-full bg-green-600 text-white py-3 px-4 rounded-lg font-semibold hover:bg-green-700 transition-colors"

              >

                Got It!

              </button>

            </div>

          </Modal>

        )}

      </div>

    </AppContext.Provider>

  );

};

export default App;