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

import { initializeApp } from 'firebase/app';

import { getAuth, signInWithCustomToken, signInAnonymously, onAuthStateChanged, createUserWithEmailAndPassword, signInWithEmailAndPassword, signOut } from 'firebase/auth';

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

import { ShoppingCart, User, LogOut, LogIn, PlusCircle, MinusCircle, Package, Receipt, Users } from 'lucide-react';

// --- Firebase Configuration and Context ---

// Global variables provided by the Canvas environment

const appId = typeof \_\_app\_id !== 'undefined' ? \_\_app\_id : 'default-app-id';

const firebaseConfig = typeof \_\_firebase\_config !== 'undefined' ? JSON.parse(\_\_firebase\_config) : {};

const initialAuthToken = typeof \_\_initial\_auth\_token !== 'undefined' ? \_\_initial\_auth\_token : null;

// Initialize Firebase

const app = initializeApp(firebaseConfig);

const auth = getAuth(app);

const db = getFirestore(app);

// Context for Auth and Firestore

const AuthContext = createContext(null);

const FirestoreContext = createContext(null);

const CartContext = createContext(null); // Context for the shopping cart

// Firestore paths

const getProductsCollectionPath = () => `/artifacts/${appId}/public/data/products`;

const getUsersProfileCollectionPath = (uid) => `/artifacts/${appId}/users/${uid}/user\_profiles`;

const getOrdersCollectionPath = () => `/artifacts/${appId}/public/data/orders`;

// --- Utility Components ---

// Simple Modal for alerts and confirmations

const Modal = ({ show, title, message, onConfirm, onCancel, confirmText = 'Aceptar', cancelText = 'Cancelar' }) => {

  if (!show) return null;

  return (

    <div className="fixed inset-0 bg-gray-600 bg-opacity-50 flex items-center justify-center p-4 z-50">

      <div className="bg-white rounded-lg shadow-xl p-6 w-full max-w-sm">

        <h3 className="text-lg font-semibold text-gray-900 mb-4">{title}</h3>

        <p className="text-gray-700 mb-6">{message}</p>

        <div className="flex justify-end space-x-3">

          {onCancel && (

            <button

              onClick={onCancel}

              className="px-4 py-2 bg-gray-200 text-gray-800 rounded-lg hover:bg-gray-300 transition-colors"

            >

              {cancelText}

            </button>

          )}

          {onConfirm && (

            <button

              onClick={onConfirm}

              className="px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 transition-colors"

            >

              {confirmText}

            </button>

          )}

        </div>

      </div>

    </div>

  );

};

// --- App Providers (Wrapping the entire application) ---

function AppProviders({ children }) {

  const [user, setUser] = useState(null);

  const [loadingAuth, setLoadingAuth] = useState(true);

  const [cart, setCart] = useState([]);

  const [showModal, setShowModal] = useState(false);

  const [modalContent, setModalContent] = useState({});

  useEffect(() => {

    // Initial Firebase Auth

    const signIn = async () => {

      try {

        if (initialAuthToken) {

          await signInWithCustomToken(auth, initialAuthToken);

        } else {

          await signInAnonymously(auth);

        }

      } catch (error) {

        console.error("Error signing in:", error);

        setModalContent({

          title: "Error de Autenticación",

          message: `Hubo un problema al iniciar sesión. ${error.message}`,

          onConfirm: () => setShowModal(false)

        });

        setShowModal(true);

      } finally {

        setLoadingAuth(false);

      }

    };

    signIn();

    // Listen for auth state changes

    const unsubscribe = onAuthStateChanged(auth, async (currentUser) => {

      setUser(currentUser);

      if (currentUser && currentUser.uid) {

        // Fetch user profile to check for isAdmin status

        const userProfileRef = doc(db, getUsersProfileCollectionPath(currentUser.uid), currentUser.uid);

        const userProfileSnap = await getDoc(userProfileRef);

        if (userProfileSnap.exists()) {

          setUser({ ...currentUser, ...userProfileSnap.data() });

        } else {

          // Create a basic profile if not exists (for non-admin users)

          await setDoc(userProfileRef, { email: currentUser.email || 'anon@example.com', name: currentUser.displayName || 'Anónimo', isAdmin: false }, { merge: true });

          setUser({ ...currentUser, email: currentUser.email || 'anon@example.com', name: currentUser.displayName || 'Anónimo', isAdmin: false });

        }

      }

      setLoadingAuth(false);

    });

    return () => unsubscribe();

  }, [initialAuthToken]);

  const addToCart = (product) => {

    setCart((prevCart) => {

      const existingItem = prevCart.find((item) => item.id === product.id);

      if (existingItem) {

        return prevCart.map((item) =>

          item.id === product.id ? { ...item, quantity: item.quantity + 1 } : item

        );

      } else {

        return [...prevCart, { ...product, quantity: 1 }];

      }

    });

    setModalContent({

      title: "Producto Añadido",

      message: `${product.name} ha sido añadido al carrito.`,

      onConfirm: () => setShowModal(false)

    });

    setShowModal(true);

  };

  const updateCartQuantity = (productId, newQuantity) => {

    setCart((prevCart) => {

      if (newQuantity <= 0) {

        return prevCart.filter((item) => item.id !== productId);

      }

      return prevCart.map((item) =>

        item.id === productId ? { ...item, quantity: newQuantity } : item

      );

    });

  };

  const clearCart = () => {

    setCart([]);

  };

  const openModal = (title, message, onConfirm = null, onCancel = null, confirmText = 'Aceptar', cancelText = 'Cancelar') => {

    setModalContent({ title, message, onConfirm, onCancel, confirmText, cancelText });

    setShowModal(true);

  };

  const closeModal = () => setShowModal(false);

  if (loadingAuth) {

    return (

      <div className="flex items-center justify-center min-h-screen bg-gray-100">

        <p className="text-xl text-gray-700">Cargando aplicación...</p>

      </div>

    );

  }

  return (

    <AuthContext.Provider value={{ user, auth, openModal, closeModal }}>

      <FirestoreContext.Provider value={{ db }}>

        <CartContext.Provider value={{ cart, addToCart, updateCartQuantity, clearCart }}>

          {children}

          <Modal

            show={showModal}

            title={modalContent.title}

            message={modalContent.message}

            onConfirm={modalContent.onConfirm}

            onCancel={modalContent.onCancel}

            confirmText={modalContent.confirmText}

            cancelText={modalContent.cancelText}

          />

        </CartContext.Provider>

      </FirestoreContext.Provider>

    </AuthContext.Provider>

  );

}

// --- Main Application Component ---

export default function App() {

  const [currentPage, setCurrentPage] = useState('home');

  const { user, openModal, closeModal } = useContext(AuthContext);

  const { cart } = useContext(CartContext);

  const handleLogout = async () => {

    openModal(

      "Confirmar Cierre de Sesión",

      "¿Estás seguro de que quieres cerrar tu sesión?",

      async () => {

        try {

          await signOut(auth);

          closeModal();

          setCurrentPage('home');

        } catch (error) {

          console.error("Error logging out:", error);

          openModal("Error", "No se pudo cerrar la sesión.");

        }

      },

      closeModal

    );

  };

  const totalCartItems = cart.reduce((acc, item) => acc + item.quantity, 0);

  return (

    <AppProviders>

      <div className="min-h-screen bg-gray-50 font-inter antialiased">

        <Navbar setCurrentPage={setCurrentPage} totalCartItems={totalCartItems} handleLogout={handleLogout} />

        <main className="container mx-auto px-4 py-8">

          {currentPage === 'home' && <HomePage setCurrentPage={setCurrentPage} />}

          {currentPage === 'cart' && <CartPage setCurrentPage={setCurrentPage} />}

          {currentPage === 'login' && <LoginPage setCurrentPage={setCurrentPage} />}

          {currentPage === 'register' && <RegisterPage setCurrentPage={setCurrentPage} />}

          {user && user.isAdmin && currentPage === 'admin' && <AdminPage />}

          {user && !user.isAdmin && currentPage === 'profile' && <UserProfilePage />}

          {/\* Fallback for non-admin user trying to access admin \*/}

          {currentPage === 'admin' && !user?.isAdmin && user && (

            <div className="text-center py-10">

              <h2 className="text-2xl font-bold text-red-600">Acceso Denegado</h2>

              <p className="text-gray-700 mt-2">No tienes permisos para ver esta página.</p>

              <button

                onClick={() => setCurrentPage('home')}

                className="mt-6 px-6 py-3 bg-blue-600 text-white rounded-lg shadow hover:bg-blue-700 transition-colors"

              >

                Volver al inicio

              </button>

            </div>

          )}

        </main>

      </div>

    </AppProviders>

  );

}

// --- Navbar Component ---

function Navbar({ setCurrentPage, totalCartItems, handleLogout }) {

  const { user } = useContext(AuthContext);

  return (

    <nav className="bg-gradient-to-r from-blue-600 to-purple-700 shadow-lg p-4 rounded-b-lg">

      <div className="container mx-auto flex justify-between items-center">

        <h1

          onClick={() => setCurrentPage('home')}

          className="text-white text-3xl font-bold tracking-tight cursor-pointer hover:text-blue-200 transition-colors"

        >

          ArtShop

        </h1>

        <div className="flex items-center space-x-6">

          <button

            onClick={() => setCurrentPage('cart')}

            className="relative p-2 rounded-full bg-blue-500 hover:bg-blue-400 transition-colors text-white focus:outline-none focus:ring-2 focus:ring-blue-300"

          >

            <ShoppingCart size={20} />

            {totalCartItems > 0 && (

              <span className="absolute -top-1 -right-1 bg-red-500 text-white text-xs font-bold rounded-full h-5 w-5 flex items-center justify-center">

                {totalCartItems}

              </span>

            )}

          </button>

          {user && user.email ? (

            <>

              {user.isAdmin && (

                <button

                  onClick={() => setCurrentPage('admin')}

                  className="px-4 py-2 bg-blue-500 text-white rounded-lg shadow-md hover:bg-blue-400 transition-colors flex items-center space-x-2"

                >

                  <Users size={18} />

                  <span>Admin</span>

                </button>

              )}

              {!user.isAdmin && (

                <button

                  onClick={() => setCurrentPage('profile')}

                  className="px-4 py-2 bg-blue-500 text-white rounded-lg shadow-md hover:bg-blue-400 transition-colors flex items-center space-x-2"

                >

                  <User size={18} />

                  <span>{user.email.split('@')[0]}</span>

                </button>

              )}

              <button

                onClick={handleLogout}

                className="px-4 py-2 bg-red-500 text-white rounded-lg shadow-md hover:bg-red-400 transition-colors flex items-center space-x-2"

              >

                <LogOut size={18} />

                <span>Salir</span>

              </button>

            </>

          ) : (

            <>

              <button

                onClick={() => setCurrentPage('login')}

                className="px-4 py-2 bg-blue-500 text-white rounded-lg shadow-md hover:bg-blue-400 transition-colors flex items-center space-x-2"

              >

                <LogIn size={18} />

                <span>Ingresar</span>

              </button>

              <button

                onClick={() => setCurrentPage('register')}

                className="px-4 py-2 bg-purple-500 text-white rounded-lg shadow-md hover:bg-purple-400 transition-colors"

              >

                Registrarse

              </button>

            </>

          )}

        </div>

      </div>

    </nav>

  );

}

// --- Product Card Component ---

function ProductCard({ product }) {

  const { addToCart } = useContext(CartContext);

  return (

    <div className="bg-white rounded-lg shadow-lg overflow-hidden flex flex-col transform hover:scale-105 transition-transform duration-300">

      <img

        src={product.imageUrl || `https://placehold.co/400x300/e0e7ff/4338ca?text=${product.name.replace(/\s/g, '+')}`}

        alt={product.name}

        className="w-full h-48 object-cover object-center"

        onError={(e) => { e.target.onerror = null; e.target.src = `https://placehold.co/400x300/e0e7ff/4338ca?text=${product.name.replace(/\s/g, '+')}` }}

      />

      <div className="p-5 flex-grow flex flex-col justify-between">

        <div>

          <h3 className="text-xl font-semibold text-gray-900 mb-2">{product.name}</h3>

          <p className="text-gray-700 text-sm mb-3 line-clamp-3">{product.description}</p>

        </div>

        <div className="flex items-center justify-between mt-4">

          <span className="text-2xl font-bold text-blue-600">${product.price.toFixed(2)}</span>

          {product.stock > 0 ? (

            <button

              onClick={() => addToCart(product)}

              className="px-5 py-2 bg-blue-600 text-white rounded-full shadow-lg hover:bg-blue-700 transition-colors duration-300 flex items-center space-x-2"

            >

              <ShoppingCart size={18} />

              <span>Añadir</span>

            </button>

          ) : (

            <span className="px-5 py-2 bg-gray-400 text-white rounded-full">Agotado</span>

          )}

        </div>

        {product.stock <= 5 && product.stock > 0 && (

            <p className="text-sm text-red-500 mt-2">¡Solo quedan {product.stock} unidades!</p>

        )}

      </div>

    </div>

  );

}

// --- Home Page Component ---

function HomePage({ setCurrentPage }) {

  const { db } = useContext(FirestoreContext);

  const { openModal } = useContext(AuthContext);

  const [products, setProducts] = useState([]);

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

  const [error, setError] = useState(null);

  useEffect(() => {

    const productsCollectionRef = collection(db, getProductsCollectionPath());

    const unsubscribe = onSnapshot(productsCollectionRef, (snapshot) => {

      const productsData = snapshot.docs.map((doc) => ({

        id: doc.id,

        ...doc.data(),

      }));

      setProducts(productsData);

      setLoading(false);

    }, (err) => {

      console.error("Error fetching products:", err);

      setError("No se pudieron cargar los productos.");

      setLoading(false);

      openModal("Error", "No se pudieron cargar los productos. Inténtalo de nuevo más tarde.");

    });

    return () => unsubscribe();

  }, [db, openModal]);

  if (loading) {

    return (

      <div className="text-center py-10">

        <p className="text-xl text-gray-700">Cargando productos...</p>

      </div>

    );

  }

  if (error) {

    return (

      <div className="text-center py-10">

        <p className="text-xl text-red-600">{error}</p>

      </div>

    );

  }

  return (

    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-8">

      {products.length > 0 ? (

        products.map((product) => (

          <ProductCard key={product.id} product={product} />

        ))

      ) : (

        <div className="md:col-span-4 text-center py-10 bg-white rounded-lg shadow-md">

          <p className="text-xl text-gray-700">No hay productos disponibles por el momento.</p>

          <p className="text-gray-500 mt-2">Visita la página de administración para agregar productos.</p>

          <button

            onClick={() => setCurrentPage('admin')}

            className="mt-6 px-6 py-3 bg-blue-600 text-white rounded-lg shadow hover:bg-blue-700 transition-colors"

          >

            Ir a Administración

          </button>

        </div>

      )}

    </div>

  );

}

// --- Cart Page Component ---

function CartPage({ setCurrentPage }) {

  const { cart, updateCartQuantity, clearCart } = useContext(CartContext);

  const { user, openModal } = useContext(AuthContext);

  const { db } = useContext(FirestoreContext);

  const total = cart.reduce((sum, item) => sum + item.price \* item.quantity, 0);

  const handleCheckout = async () => {

    if (!user || !user.email) {

      openModal(

        "Iniciar Sesión Requerido",

        "Debes iniciar sesión para completar tu compra.",

        () => setCurrentPage('login'),

        null,

        "Ir a Iniciar Sesión"

      );

      return;

    }

    if (cart.length === 0) {

      openModal("Carrito Vacío", "Tu carrito está vacío. Añade algunos productos antes de comprar.");

      return;

    }

    openModal(

      "Confirmar Compra",

      `¿Confirmas la compra de ${cart.reduce((acc, item) => acc + item.quantity, 0)} artículos por un total de $${total.toFixed(2)}?`,

      async () => {

        try {

          const batchUpdates = [];

          const productsCollectionRef = collection(db, getProductsCollectionPath());

          const ordersCollectionRef = collection(db, getOrdersCollectionPath());

          // Check stock before processing

          for (const item of cart) {

            const productRef = doc(productsCollectionRef, item.id);

            const productSnap = await getDoc(productRef);

            if (!productSnap.exists() || productSnap.data().stock < item.quantity) {

              openModal("Error de Stock", `No hay suficiente stock de ${item.name}. Disponible: ${productSnap.data()?.stock || 0}`);

              closeModal();

              return; // Stop checkout if stock is insufficient

            }

          }

          // Process order and update stock

          for (const item of cart) {

            const productRef = doc(productsCollectionRef, item.id);

            batchUpdates.push(updateDoc(productRef, { stock: increment(-item.quantity) })); // Use increment for atomic updates

          }

          const order = {

            userId: user.uid,

            customerName: user.displayName || user.email,

            customerEmail: user.email,

            items: cart.map(item => ({

              productId: item.id,

              name: item.name,

              quantity: item.quantity,

              price: item.price,

            })),

            totalAmount: total,

            orderDate: new Date(),

            status: 'pending',

          };

          batchUpdates.push(addDoc(ordersCollectionRef, order));

          // Execute all batch updates (stock and order)

          // For simplicity, we are not using a Firestore transaction here, but it would be ideal for atomicity.

          // For now, we'll just run them sequentially or in parallel.

          await Promise.all(batchUpdates);

          openModal("Compra Exitosa", "Tu compra ha sido realizada con éxito. ¡Gracias!", () => {

            clearCart();

            setCurrentPage('home');

            closeModal();

          });

        } catch (error) {

          console.error("Error al procesar la compra:", error);

          openModal("Error de Compra", `No se pudo completar la compra: ${error.message}`);

        }

      },

      closeModal

    );

  };

  // Helper for Firestore increment (not directly available in client-side, needs to be done via updateDoc)

  // This is a placeholder for the actual updateDoc behavior. Firestore updateDoc handles atomic increments.

  const increment = (value) => value;

  return (

    <div className="bg-white rounded-lg shadow-xl p-8 max-w-3xl mx-auto">

      <h2 className="text-3xl font-bold text-gray-900 mb-6 text-center">Tu Carrito de Compras</h2>

      {cart.length === 0 ? (

        <div className="text-center py-10">

          <p className="text-xl text-gray-700 mb-4">Tu carrito está vacío.</p>

          <button

            onClick={() => setCurrentPage('home')}

            className="px-6 py-3 bg-blue-600 text-white rounded-lg shadow hover:bg-blue-700 transition-colors"

          >

            Explorar Productos

          </button>

        </div>

      ) : (

        <>

          <div className="space-y-6 mb-8">

            {cart.map((item) => (

              <div key={item.id} className="flex items-center justify-between p-4 bg-gray-50 rounded-lg shadow-sm">

                <div className="flex items-center space-x-4">

                  <img

                    src={item.imageUrl || `https://placehold.co/80x60/e0e7ff/4338ca?text=${item.name.replace(/\s/g, '+')}`}

                    alt={item.name}

                    className="w-20 h-16 object-cover rounded-md"

                    onError={(e) => { e.target.onerror = null; e.target.src = `https://placehold.co/80x60/e0e7ff/4338ca?text=${item.name.replace(/\s/g, '+')}` }}

                  />

                  <div>

                    <h3 className="text-lg font-semibold text-gray-900">{item.name}</h3>

                    <p className="text-blue-600 font-medium">${item.price.toFixed(2)}</p>

                  </div>

                </div>

                <div className="flex items-center space-x-3">

                  <button

                    onClick={() => updateCartQuantity(item.id, item.quantity - 1)}

                    className="p-2 rounded-full bg-gray-200 text-gray-700 hover:bg-gray-300 transition-colors"

                  >

                    <MinusCircle size={18} />

                  </button>

                  <span className="text-lg font-semibold text-gray-900 w-8 text-center">{item.quantity}</span>

                  <button

                    onClick={() => updateCartQuantity(item.id, item.quantity + 1)}

                    className="p-2 rounded-full bg-blue-500 text-white hover:bg-blue-600 transition-colors"

                  >

                    <PlusCircle size={18} />

                  </button>

                </div>

              </div>

            ))}

          </div>

          <div className="flex justify-between items-center border-t border-gray-200 pt-6 mt-6">

            <h3 className="text-xl font-bold text-gray-900">Total:</h3>

            <span className="text-3xl font-bold text-blue-700">${total.toFixed(2)}</span>

          </div>

          <div className="flex justify-between mt-8 space-x-4">

            <button

              onClick={clearCart}

              className="px-6 py-3 bg-red-500 text-white rounded-lg shadow hover:bg-red-600 transition-colors flex-1"

            >

              Vaciar Carrito

            </button>

            <button

              onClick={handleCheckout}

              className="px-6 py-3 bg-green-600 text-white rounded-lg shadow hover:bg-green-700 transition-colors flex-1"

            >

              Finalizar Compra

            </button>

          </div>

        </>

      )}

    </div>

  );

}

// --- Login Page Component ---

function LoginPage({ setCurrentPage }) {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const { auth, openModal, closeModal } = useContext(AuthContext);

  const handleLogin = async (e) => {

    e.preventDefault();

    try {

      await signInWithEmailAndPassword(auth, email, password);

      openModal("Inicio de Sesión Exitoso", "¡Bienvenido de nuevo!", () => {

        setCurrentPage('home');

        closeModal();

      });

    } catch (error) {

      console.error("Error logging in:", error);

      openModal("Error de Inicio de Sesión", `Credenciales incorrectas: ${error.message}`);

    }

  };

  return (

    <div className="max-w-md mx-auto bg-white rounded-lg shadow-xl p-8 mt-10">

      <h2 className="text-3xl font-bold text-gray-900 mb-6 text-center">Ingresar</h2>

      <form onSubmit={handleLogin} className="space-y-5">

        <div>

          <label htmlFor="email" className="block text-gray-700 text-sm font-semibold mb-2">

            Email

          </label>

          <input

            type="email"

            id="email"

            value={email}

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

            className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-blue-500 focus:border-blue-500"

            placeholder="tu@email.com"

            required

          />

        </div>

        <div>

          <label htmlFor="password" className="block text-gray-700 text-sm font-semibold mb-2">

            Contraseña

          </label>

          <input

            type="password"

            id="password"

            value={password}

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

            className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-blue-500 focus:border-blue-500"

            placeholder="\*\*\*\*\*\*\*\*"

            required

          />

        </div>

        <button

          type="submit"

          className="w-full py-3 bg-blue-600 text-white rounded-lg shadow-md hover:bg-blue-700 transition-colors font-semibold"

        >

          Ingresar

        </button>

      </form>

      <p className="text-center text-gray-600 mt-6">

        ¿No tienes una cuenta?{' '}

        <button onClick={() => setCurrentPage('register')} className="text-blue-600 hover:underline">

          Regístrate aquí

        </button>

      </p>

    </div>

  );

}

// --- Register Page Component ---

function RegisterPage({ setCurrentPage }) {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const [confirmPassword, setConfirmPassword] = useState('');

  const { auth, openModal, closeModal } = useContext(AuthContext);

  const { db } = useContext(FirestoreContext);

  const handleRegister = async (e) => {

    e.preventDefault();

    if (password !== confirmPassword) {

      openModal("Error de Registro", "Las contraseñas no coinciden.");

      return;

    }

    try {

      const userCredential = await createUserWithEmailAndPassword(auth, email, password);

      const user = userCredential.user;

      // Create user profile in Firestore

      const userProfileRef = doc(db, getUsersProfileCollectionPath(user.uid), user.uid);

      await setDoc(userProfileRef, { email: user.email, name: user.displayName || 'Cliente', isAdmin: false });

      openModal("Registro Exitoso", "¡Tu cuenta ha sido creada! Por favor, inicia sesión.", () => {

        setCurrentPage('login');

        closeModal();

      });

    } catch (error) {

      console.error("Error registering:", error);

      openModal("Error de Registro", `No se pudo crear la cuenta: ${error.message}`);

    }

  };

  return (

    <div className="max-w-md mx-auto bg-white rounded-lg shadow-xl p-8 mt-10">

      <h2 className="text-3xl font-bold text-gray-900 mb-6 text-center">Registrarse</h2>

      <form onSubmit={handleRegister} className="space-y-5">

        <div>

          <label htmlFor="email" className="block text-gray-700 text-sm font-semibold mb-2">

            Email

          </label>

          <input

            type="email"

            id="email"

            value={email}

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

            className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-blue-500 focus:border-blue-500"

            placeholder="tu@email.com"

            required

          />

        </div>

        <div>

          <label htmlFor="password" className="block text-gray-700 text-sm font-semibold mb-2">

            Contraseña

          </label>

          <input

            type="password"

            id="password"

            value={password}

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

            className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-blue-500 focus:border-blue-500"

            placeholder="\*\*\*\*\*\*\*\*"

            required

          />

        </div>

        <div>

          <label htmlFor="confirmPassword" className="block text-gray-700 text-sm font-semibold mb-2">

            Confirmar Contraseña

          </label>

          <input

            type="password"

            id="confirmPassword"

            value={confirmPassword}

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

            className="w-full px-4 py-2 border border-gray-300 rounded-lg focus:ring-blue-500 focus:border-blue-500"

            placeholder="\*\*\*\*\*\*\*\*"

            required

          />

        </div>

        <button

          type="submit"

          className="w-full py-3 bg-purple-600 text-white rounded-lg shadow-md hover:bg-purple-700 transition-colors font-semibold"

        >

          Registrarse

        </button>

      </form>

      <p className="text-center text-gray-600 mt-6">

        ¿Ya tienes una cuenta?{' '}

        <button onClick={() => setCurrentPage('login')} className="text-blue-600 hover:underline">

          Ingresar aquí

        </button>

      </p>

    </div>

  );

}

// --- User Profile Page Component (for non-admin users) ---

function UserProfilePage() {

  const { user, openModal } = useContext(AuthContext);

  const { db } = useContext(FirestoreContext);

  const [userOrders, setUserOrders] = useState([]);

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

  const [error, setError] = useState(null);

  useEffect(() => {

    if (!user || !user.uid) return;

    const ordersCollectionRef = collection(db, getOrdersCollectionPath());

    const q = query(ordersCollectionRef, where("userId", "==", user.uid));

    const unsubscribe = onSnapshot(q, (snapshot) => {

      const ordersData = snapshot.docs.map((doc) => ({

        id: doc.id,

        ...doc.data(),

      }));

      // Sort orders by date descending

      ordersData.sort((a, b) => b.orderDate.toDate() - a.orderDate.toDate());

      setUserOrders(ordersData);

      setLoading(false);

    }, (err) => {

      console.error("Error fetching user orders:", err);

      setError("No se pudieron cargar tus órdenes.");

      setLoading(false);

      openModal("Error", "No se pudieron cargar tus órdenes. Inténtalo de nuevo más tarde.");

    });

    return () => unsubscribe();

  }, [user, db, openModal]);

  if (loading) {

    return (

      <div className="text-center py-10">

        <p className="text-xl text-gray-700">Cargando tus órdenes...</p>

      </div>

    );

  }

  if (error) {

    return (

      <div className="text-center py-10">

        <p className="text-xl text-red-600">{error}</p>

      </div>

    );

  }

  return (

    <div className="max-w-4xl mx-auto bg-white rounded-lg shadow-xl p-8 mt-10">

      <h2 className="text-3xl font-bold text-gray-900 mb-6 text-center">Mi Perfil</h2>

      <p className="text-lg text-gray-700 mb-4">

        <span className="font-semibold">Email:</span> {user.email}

      </p>

      <h3 className="text-2xl font-bold text-gray-800 mt-8 mb-4 border-t pt-6">Mis Compras</h3>

      {userOrders.length === 0 ? (

        <p className="text-gray-600">Aún no has realizado ninguna compra.</p>

      ) : (

        <div className="space-y-6">

          {userOrders.map((order) => (

            <div key={order.id} className="bg-gray-50 rounded-lg shadow-sm p-5 border-l-4 border-blue-500">

              <div className="flex justify-between items-center mb-3">

                <p className="font-semibold text-gray-800">Orden ID: <span className="text-blue-700">{order.id}</span></p>

                <p className={`px-3 py-1 text-sm font-semibold rounded-full ${

                  order.status === 'completed' ? 'bg-green-100 text-green-800' :

                  order.status === 'shipped' ? 'bg-indigo-100 text-indigo-800' :

                  'bg-yellow-100 text-yellow-800'

                }`}>

                  {order.status.charAt(0).toUpperCase() + order.status.slice(1)}

                </p>

              </div>

              <p className="text-sm text-gray-600 mb-2">Fecha: {new Date(order.orderDate.toDate()).toLocaleString()}</p>

              <ul className="space-y-2 mb-3">

                {order.items.map((item, index) => (

                  <li key={index} className="text-gray-700 text-sm">

                    {item.name} x {item.quantity} - ${item.price.toFixed(2)} c/u

                  </li>

                ))}

              </ul>

              <p className="text-lg font-bold text-blue-700 border-t pt-2 mt-2">Total: ${order.totalAmount.toFixed(2)}</p>

            </div>

          ))}

        </div>

      )}

    </div>

  );

}

// --- Admin Page Component ---

function AdminPage() {

  const { db } = useContext(FirestoreContext);

  const { openModal, closeModal } = useContext(AuthContext);

  const [currentTab, setCurrentTab] = useState('products'); // 'products', 'orders', 'users'

  // Product states

  const [products, setProducts] = useState([]);

  const [newProductName, setNewProductName] = useState('');

  const [newProductDescription, setNewProductDescription] = useState('');

  const [newProductPrice, setNewProductPrice] = useState('');

  const [newProductStock, setNewProductStock] = useState('');

  const [newProductImageUrl, setNewProductImageUrl] = useState('');

  const [editingProduct, setEditingProduct] = useState(null); // Product being edited

  // Order states

  const [orders, setOrders] = useState([]);

  // User states

  const [appUsers, setAppUsers] = useState([]); // Renamed to avoid conflict with `user` from context

  // Listen for Products

  useEffect(() => {

    const productsCollectionRef = collection(db, getProductsCollectionPath());

    const unsubscribe = onSnapshot(productsCollectionRef, (snapshot) => {

      const productsData = snapshot.docs.map((doc) => ({

        id: doc.id,

        ...doc.data(),

      }));

      setProducts(productsData);

    });

    return () => unsubscribe();

  }, [db]);

  // Listen for Orders

  useEffect(() => {

    const ordersCollectionRef = collection(db, getOrdersCollectionPath());

    const unsubscribe = onSnapshot(ordersCollectionRef, (snapshot) => {

      const ordersData = snapshot.docs.map((doc) => ({

        id: doc.id,

        ...doc.data(),

        orderDate: doc.data().orderDate.toDate(), // Convert Firestore Timestamp to Date object

      }));

      // Sort orders by date descending

      ordersData.sort((a, b) => b.orderDate - a.orderDate);

      setOrders(ordersData);

    });

    return () => unsubscribe();

  }, [db]);

  // Listen for Users

  useEffect(() => {

    const fetchUsers = async () => {

      const usersRef = collection(db, `/artifacts/${appId}/users`);

      const userDocs = await getDocs(usersRef);

      const fetchedUsers = [];

      for (const userDoc of userDocs.docs) {

        const userProfileRef = doc(db, getUsersProfileCollectionPath(userDoc.id), userDoc.id);

        const userProfileSnap = await getDoc(userProfileRef);

        if (userProfileSnap.exists()) {

          fetchedUsers.push({ id: userDoc.id, ...userProfileSnap.data() });

        }

      }

      setAppUsers(fetchedUsers);

    };

    // This fetch will run once or on demand. For real-time, it would be more complex

    // as user profiles are in subcollections.

    fetchUsers();

  }, [db]);

  // --- Product Management Handlers ---

  const handleAddOrUpdateProduct = async (e) => {

    e.preventDefault();

    const productsCollectionRef = collection(db, getProductsCollectionPath());

    const productData = {

      name: newProductName,

      description: newProductDescription,

      price: parseFloat(newProductPrice),

      stock: parseInt(newProductStock, 10),

      imageUrl: newProductImageUrl || '',

    };

    try {

      if (editingProduct) {

        await updateDoc(doc(productsCollectionRef, editingProduct.id), productData);

        openModal("Producto Actualizado", `"${productData.name}" ha sido actualizado.`);

      } else {

        await addDoc(productsCollectionRef, productData);

        openModal("Producto Añadido", `"${productData.name}" ha sido añadido.`);

      }

      // Reset form

      setNewProductName('');

      setNewProductDescription('');

      setNewProductPrice('');

      setNewProductStock('');

      setNewProductImageUrl('');

      setEditingProduct(null);

    } catch (error) {

      console.error("Error adding/updating product:", error);

      openModal("Error", "No se pudo añadir/actualizar el producto.");

    }

  };

  const handleEditProduct = (product) => {

    setEditingProduct(product);

    setNewProductName(product.name);

    setNewProductDescription(product.description);

    setNewProductPrice(product.price);

    setNewProductStock(product.stock);

    setNewProductImageUrl(product.imageUrl || '');

    setCurrentTab('products'); // Switch to products tab if not already

  };

  const handleDeleteProduct = (productId) => {

    openModal(

      "Confirmar Eliminación",

      "¿Estás seguro de que quieres eliminar este producto? Esta acción no se puede deshacer.",

      async () => {

        try {

          const productRef = doc(db, getProductsCollectionPath(), productId);

          await deleteDoc(productRef);

          openModal("Producto Eliminado", "El producto ha sido eliminado.");

          closeModal();

        } catch (error) {

          console.error("Error deleting product:", error);

          openModal("Error", "No se pudo eliminar el producto.");

        }

      },

      closeModal

    );

  };

  // --- Order Management Handlers ---

  const handleUpdateOrderStatus = (orderId, newStatus) => {

    openModal(

      "Confirmar Actualización",

      `¿Estás seguro de que quieres cambiar el estado de la orden ${orderId} a "${newStatus}"?`,

      async () => {

        try {

          const orderRef = doc(db, getOrdersCollectionPath(), orderId);

          await updateDoc(orderRef, { status: newStatus });

          openModal("Estado Actualizado", `El estado de la orden ${orderId} ha sido actualizado a "${newStatus}".`);

          closeModal();

        } catch (error) {

          console.error("Error updating order status:", error);

          openModal("Error", "No se pudo actualizar el estado de la orden.");

        }

      },

      closeModal

    );

  };

  // --- Admin User Management Handlers ---

  const handleToggleAdminStatus = async (userId, currentIsAdmin) => {

    openModal(

      "Confirmar Cambio de Rol",

      `¿Estás seguro de que quieres ${currentIsAdmin ? 'quitar' : 'otorgar'} permisos de administrador a este usuario?`,

      async () => {

        try {

          const userProfileRef = doc(db, getUsersProfileCollectionPath(userId), userId);

          await updateDoc(userProfileRef, { isAdmin: !currentIsAdmin });

          openModal("Rol Actualizado", `El rol del usuario ha sido actualizado.`);

          closeModal();

        } catch (error) {

          console.error("Error updating user admin status:", error);

          openModal("Error", "No se pudo actualizar el rol del usuario.");

        }

      },

      closeModal

    );

  };

  return (

    <div className="max-w-6xl mx-auto bg-white rounded-lg shadow-xl p-8 mt-10">

      <h2 className="text-3xl font-bold text-gray-900 mb-6 text-center">Panel de Administración</h2>

      {/\* Tabs Navigation \*/}

      <div className="flex justify-center border-b border-gray-200 mb-8">

        <button

          onClick={() => setCurrentTab('products')}

          className={`py-3 px-6 text-lg font-medium ${

            currentTab === 'products' ? 'text-blue-600 border-b-2 border-blue-600' : 'text-gray-600 hover:text-gray-800'

          }`}

        >

          <Package className="inline-block mr-2" size={20} />Productos

        </button>

        <button

          onClick={() => setCurrentTab('orders')}

          className={`py-3 px-6 text-lg font-medium ${

            currentTab === 'orders' ? 'text-blue-600 border-b-2 border-blue-600' : 'text-gray-600 hover:text-gray-800'

          }`}

        >

          <Receipt className="inline-block mr-2" size={20} />Ventas

        </button>

        <button

          onClick={() => setCurrentTab('users')}

          className={`py-3 px-6 text-lg font-medium ${

            currentTab === 'users' ? 'text-blue-600 border-b-2 border-blue-600' : 'text-gray-600 hover:text-gray-800'

          }`}

        >

          <Users className="inline-block mr-2" size={20} />Clientes

        </button>

      </div>

      {/\* Products Tab Content \*/}

      {currentTab === 'products' && (

        <div className="grid grid-cols-1 md:grid-cols-2 gap-8">