ChatGPT

NovaPOS Cashier POS Screen Implementation

Below is a complete set of React + TypeScript files implementing the NovaPOS Cashier POS screen and logic. The code is structured for integration into the `frontends/pos-app` directory, using Tailwind CSS for styling and React Router for navigation. Each file is described with its path and purpose.

Pages

`frontends/pos-app/src/pages/CashierPage.tsx`

```
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { useAuth } from '../AuthContext';
import { useCart } from '../CartContext';
import { useOrders } from '../OrderContext';
import type { PaymentMethod } from '../OrderContext';
import { useProducts } from '../hooks/useProducts';
import { useOfflineQueue } from '../hooks/useOfflineQueue';
import { useSyncOnReconnect } from '../hooks/useSyncOnReconnect';
import OfflineBanner from '../components/pos/OfflineBanner';
import QueuedOrdersBanner from '../components/pos/QueuedOrdersBanner';
import SearchBar from '../components/pos/SearchBar';
import CategoryFilter from '../components/pos/CategoryFilter';
import ProductGrid from '../components/pos/ProductGrid';
import CartSidebar from '../components/pos/CartSidebar';
import SubmitSalePanel from '../components/pos/SubmitSalePanel';
import RecentOrdersDrawer from '../components/pos/RecentOrdersDrawer';
import ReplaceItemModal from '../components/pos/ReplaceItemModal';

const CashierPage: React.FC = () => {
  const navigate = useNavigate();
  const { isLoggedIn, logout } = useAuth();
  const { cart, addItem, removeItem, incrementItemQuantity,
decrementItemQuantity, clearCart, totalAmount } = useCart();
  const { isOnline, queuedOrders, isSyncing, submitOrder } = useOrders();
  const { products, isLoading: isLoadingProducts, error, isOfflineResult,
categories, reload: reloadProducts } = useProducts();
  const [query, setQuery] = useState<string>('');
  const [categoryFilter, setCategoryFilter] = useState<string>('All');
  const [drawerOpen, setDrawerOpen] = useState<boolean>(false);
  const [paymentMethod, setPaymentMethod] = useState<PaymentMethod>('cash');
  const [inactiveItems, setInactiveItems] = useState<string[]>([]);
```

```
  const [pendingSubmit, setPendingSubmit] = useState<boolean>(false);
  useSyncOnReconnect();  // Ensure offline orders sync on reconnect

  // Redirect to login if not authenticated
  useEffect(() => {
    if (!isLoggedIn) navigate('/login', { replace: true });
  }, [isLoggedIn, navigate]);

  // Filter products by search query and category
  const filteredProducts = products.filter(p => {
    const matchesQuery =
      query.trim() === '' ||
      p.name.toLowerCase().includes(query.toLowerCase()) ||
      (p.sku ?? '').toLowerCase().includes(query.toLowerCase());
    const matchesCategory =
      categoryFilter === 'All' ||
      (p.category && p.category.toLowerCase() === categoryFilter.toLowerCase())
||
      (p.description &&
p.description.toLowerCase().includes(categoryFilter.toLowerCase()));
    return matchesQuery && matchesCategory;
  });

  // Identify any cart items that are inactive (not present in current product
list)
  const inactiveCartItemIds = cart
    .filter(item => !products.find(p => p.id === item.id))
    .map(item => item.id);
  // Mark currently inactive item (for ReplaceItemModal)
  const currentInactiveId = inactiveItems.length > 0 ? inactiveItems[0] : null;
  const currentInactiveItem = currentInactiveId ? cart.find(ci => ci.id ===
currentInactiveId) : null;

  // Handle clicking the "Submit Sale" button
  const handleSubmitSale = async () => {
    if (!cart.length) return;  // no items to submit
    // Check for inactive items before submitting
    try {
      // Fetch latest catalog to verify active status of items
      const headers: Record<string, string> = {};
      const { currentUser, token } = useAuth();  // get tenant and token for
request
      const tenantId = currentUser?.tenant_id ? String(currentUser.tenant_id) :
null;
      if (tenantId) headers['X-Tenant-ID'] = tenantId;
      if (token) headers['Authorization'] = `Bearer ${token}`;
      const response = await fetch(`$
{import.meta.env.VITE_PRODUCT_SERVICE_URL ?? 'http://localhost:8081'}/
```

```
products`, { headers });
      let allProducts: any[] = [];
      if (response.ok) {
        allProducts = await response.json();  // includes inactive products
      } else {
        // If fetch fails (offline), use cached products from localStorage
        const cacheKey = `productCache:${tenantId}`;
        const cached = localStorage.getItem(cacheKey);
        allProducts = cached ? JSON.parse(cached) : [];
      }
      // Map product ID to active flag
      const activeMap: Record<string, boolean> = {};
      for (const prod of allProducts) {
        if (prod && (typeof prod.id === 'string' || typeof prod.id ===
'number')) {
          const id = typeof prod.id === 'string' ? prod.id : String(prod.id);
          activeMap[id] = prod.active !== false;
        }
      }
      // Determine which cart items are inactive
      const flaggedIds: string[] = [];
      for (const item of cart) {
        if (activeMap[item.id] === false) {
          flaggedIds.push(item.id);
        }
      }
      if (flaggedIds.length > 0) {
        // Open ReplaceItemModal for the first inactive item (sequence through
others if multiple)
        setInactiveItems(flaggedIds);
        setPendingSubmit(true);
        return;
      }
    } catch (err) {
      console.error('Error checking inactive items', err);
      // If any error in validation, we proceed with submission (assuming items
are active)
    }
    // If no inactive items, proceed to submit immediately
    triggerOrderSubmission();
  };

  // Perform the actual order submission via OrderContext
  const triggerOrderSubmission = async () => {
    const itemsPayload = cart.map(item => ({
      product_id: item.id,
      quantity: item.quantity,
      unit_price: item.price,
```

```
      line_total: Number((item.price * item.quantity).toFixed(2)),
    }));
    const payload = {
      items: itemsPayload,
      payment_method: paymentMethod,
      total: Number(totalAmount.toFixed(2)),
    };
    try {
      const result = await submitOrder(payload);
      clearCart();  // clear cart after successful submission (online or queued)
      // If order queued (offline), no further action needed (banner will show
queued status)
      if (result.status === 'submitted') {
        // If payment URL is returned for card/crypto, open it for payment
        if (result.payment && (result.payment.paymentUrl ||
result.payment.payment_url)) {
          const payUrl = result.payment.paymentUrl ??
result.payment.payment_url;
          if (payUrl) {
            window.open(payUrl, '_blank');
          }
        }
        // Optionally, could display a success notification or modal here
      }
    } catch (error) {
      console.error('Order submission failed', error);
      // TODO: Handle submission error (e.g., network error) if not already
queued by context
    }
    setPendingSubmit(false);
  };

  // Effect: if pendingSubmit is true and all inactive items have been handled,
trigger final submission
  useEffect(() => {
    if (pendingSubmit && inactiveItems.length === 0) {
      triggerOrderSubmission();
    }
  }, [pendingSubmit, inactiveItems]);

  // Handle replacement of an inactive item from modal
  const handleReplaceItem = (oldItemId: string, replacementProductId: string)
=> {
    const oldItem = cart.find(ci => ci.id === oldItemId);
    const replacement = products.find(p => p.id === replacementProductId);
    if (!oldItem || !replacement) return;
    const { id, name, price, sku } = replacement;
    // Remove the inactive item, then add the replacement product with the same
```

```
quantity
    const qty = oldItem.quantity;
    removeItem(oldItemId);
    for (let i = 0; i < qty; i++) {
      addItem({ id, name, price,
sku });  // CartContext will increment quantity if already exists
    }
    // Remove this item from the inactiveItems queue and proceed to next or
submission
    setInactiveItems(prev => prev.filter(x => x !== oldItemId));
  };

  // Handle removal of an inactive item from modal
  const handleRemoveItem = (oldItemId: string) => {
    removeItem(oldItemId);
    setInactiveItems(prev => prev.filter(x => x !== oldItemId));
  };

  // Handle cancel in ReplaceItemModal (abort submission)
  const handleCancelReplace = () => {
    setInactiveItems([]);
    setPendingSubmit(false);
  };

  return (
    <div className="min-h-screen flex flex-col bg-gray-100 dark:bg-gray-900">
      {/* Header */}
      <header className="flex items-center justify-between px-6 py-4 bg-white
dark:bg-gray-800 shadow-md">
        <div className="flex items-center gap-3">
          <img src="/assets/logo_transparent.png" alt="NovaPOS Logo"
className="h-10 w-auto" />
          <span className="text-2xl font-bold text-gray-800 dark:text-gray-100
tracking-tight">NovaPOS</span>
        </div>
        <div className="flex items-center gap-3">
          <button
            className="px-4 py-2 rounded border border-cyan-500 text-cyan-700
hover:bg-cyan-500 hover:text-white transition-colors"
            onClick={() => setDrawerOpen(true)}
          >
            Orders{queuedOrders.length > 0 ? ` (${queuedOrders.length})` : ''}
          </button>
          <button
            className="bg-red-500 text-white px-4 py-2 rounded hover:bg-red-600"
            onClick={() => { logout(); navigate('/login'); }}
          >
            Logout
```

```
          </button>
        </div>
      </header>

      {/* Offline/Queued banners */}
      {!isOnline && (
        <OfflineBanner queuedCount={queuedOrders.length} />
      )}
      {isOnline && queuedOrders.length > 0 && (
        <QueuedOrdersBanner
          count={queuedOrders.length}
          syncing={isSyncing}
          onSync={() => { /* Trigger manual retry sync */ const { retryQueue }
= useOfflineQueue(); retryQueue().catch(console.warn); }}
        />
      )}

      {/* Main content: Products + Cart/Checkout */}
      <main className="flex-1 flex flex-col md:flex-row md:items-start
md:justify-center px-4 py-6 gap-6">
        {/* Product list section */}
        <section className="flex-1 max-w-3xl">
          <div className="flex flex-col sm:flex-row sm:items-center sm:justify-
between gap-4 mb-4">
            <SearchBar query={query} onQueryChange={setQuery} />
            {categories.length > 1 && (
              <CategoryFilter categories={categories} selected={categoryFilter}
onSelect={setCategoryFilter} />
            )}
          </div>
          {isLoadingProducts && (
            <div className="text-sm text-gray-500 mb-2">Loading products...</
div>
          )}
          {isOfflineResult && !isLoadingProducts && (
            <div className="text-sm text-amber-600 mb-2">Offline mode: showing
last synced catalog.</div>
          )}
          {error && !isLoadingProducts && (
            <div className="text-sm text-red-600 mb-2">{error}</div>
          )}
          <ProductGrid products={filteredProducts} onAddProduct={prod =>
addItem({ id: prod.id, name: prod.name, price: prod.price, sku: prod.sku })} />
          {!isLoadingProducts && filteredProducts.length === 0 && (
            <div className="text-center text-gray-500 mt-4">
              No products found. Try adjusting your search or category filter.
            </div>
          )}
```

```
        </section>

        {/* Cart and checkout section */}
        <aside className="w-full max-w-md mx-auto">
          <CartSidebar
            items={cart}
            onAddQty={incrementItemQuantity}
            onSubQty={decrementItemQuantity}
            onRemoveItem={removeItem}
          />
          <SubmitSalePanel
            total={totalAmount}
            paymentMethod={paymentMethod}
            onPaymentMethodChange={setPaymentMethod}
            onSubmit={handleSubmitSale}
            submitting={false}  {/* will be controlled internally via
idempotent logic */}
          />
        </aside>
      </main>

      {/* Recent orders side-drawer */}
      <RecentOrdersDrawer open={drawerOpen} onClose={() =>
setDrawerOpen(false)} />

      {/* Modal for replacing inactive items */}
      {currentInactiveItem && (
        <ReplaceItemModal
          item={currentInactiveItem}
          products={products}
          onReplace={(replacementId) =>
handleReplaceItem(currentInactiveItem.id, replacementId)}
          onRemove={() => handleRemoveItem(currentInactiveItem.id)}
          onCancel={handleCancelReplace}
        />
      )}
    </div>
  );
};

export default CashierPage;
```

Note: Be sure to add the new route for the CashierPage in your router (e.g., in `App.tsx`):

```
<Route path="/pos" element={<CashierPage />} />
```

Components (POS)

`frontends/pos-app/src/components/pos/OfflineBanner.tsx`

```
import React from 'react';

interface OfflineBannerProps {
  queuedCount: number;
}
const OfflineBanner: React.FC<OfflineBannerProps> = ({ queuedCount }) => (
  <div className="w-full bg-amber-200 text-amber-900 px-6 py-3 text-sm text-
center">
    Offline mode – {queuedCount} order{queuedCount === 1 ? '' : 's'} queued.
Sales will sync automatically once reconnected.
  </div>
);
export default OfflineBanner;
```

`frontends/pos-app/src/components/pos/QueuedOrdersBanner.tsx`

```
import React from 'react';

interface QueuedOrdersBannerProps {
  count: number;
  syncing: boolean;
  onSync: () => void;
}
const QueuedOrdersBanner: React.FC<QueuedOrdersBannerProps> = ({ count,
syncing, onSync }) => (
  <div className="w-full bg-sky-200 text-sky-900 px-6 py-3 text-sm text-center
flex items-center justify-center">
    {syncing ? (
      <>Synchronizing queued orders...</>
    ) : (
      <>
        {count} queued order{count === 1 ? '' : 's'} awaiting sync.
        <button
          onClick={onSync}
          disabled={syncing || count === 0}
          className="ml-4 px-3 py-1 border border-sky-800 rounded text-sky-800
hover:bg-sky-300 disabled:opacity-50"
        >
          {syncing ? 'Syncing...' : 'Retry Sync'}
        </button>
      </>
```

```
    )}
  </div>
);
export default QueuedOrdersBanner;
```

frontends/pos-app/src/components/pos/SearchBar.tsx

```
import React from 'react';

interface SearchBarProps {
  query: string;
  onQueryChange: (value: string) => void;
}
const SearchBar: React.FC<SearchBarProps> = ({ query, onQueryChange }) => (
  <input
    type="search"
    value={query}
    onChange={e => onQueryChange(e.target.value)}
    placeholder="Search products..."
    className="w-full sm:w-64 px-4 py-2 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-cyan-500"
    aria-label="Search products"
  />
);
export default SearchBar;
```

frontends/pos-app/src/components/pos/CategoryFilter.tsx

```
import React from 'react';

interface CategoryFilterProps {
  categories: string[];
  selected: string;
  onSelect: (category: string) => void;
}
const CategoryFilter: React.FC<CategoryFilterProps> = ({ categories, selected,
onSelect }) => {
  if (categories.length <= 1) {
    return null;  // Hide filter if no categories (or only "All")
  }
  return (
    <select
      value={selected}
      onChange={e => onSelect(e.target.value)}
      className="px-4 py-2 border border-gray-300 rounded-lg focus:outline-none
```

```
focus:ring-2 focus:ring-cyan-500"
      aria-label="Filter by category"
    >
      {categories.map(cat => (
        <option key={cat} value={cat}>
          {cat}
        </option>
      ))}
    </select>
  );
};
export default CategoryFilter;
```

frontends/pos-app/src/components/pos/ProductGrid.tsx

```
import React from 'react';
import type { Product } from '../../hooks/useProducts';

interface ProductGridProps {
  products: Product[];
  onAddProduct: (product: Product) => void;
}
const ProductGrid: React.FC<ProductGridProps> = ({ products, onAddProduct }) =>
(
  <div className="grid grid-cols-2 sm:grid-cols-3 md:grid-cols-4 gap-4">
    {products.map(product => (
      <div
        key={product.id}
        className="flex flex-col items-center bg-white dark:bg-gray-800 rounded
shadow p-4 cursor-pointer hover:bg-gray-50 dark:hover:bg-gray-700"
      >
        <img
          src={product.image_url || '/assets/product_placeholder.png'}
          alt={product.name}
          className="w-16 h-16 object-cover mb-2"
        />
        <div className="text-center">
          <div className="text-sm font-medium text-gray-800 dark:text-
gray-100">{product.name}</div>
          <div className="text-sm text-gray-500 dark:text-gray-400">$
{product.price.toFixed(2)}</div>
        </div>
        <button
          onClick={() => onAddProduct(product)}
          className="mt-2 px-3 py-1 bg-cyan-600 text-white text-sm rounded
hover:bg-cyan-700 focus:outline-none"
```

```
        >
          Add
        </button>
      </div>
    ))}
  </div>
);
export default ProductGrid;
```

frontends/pos-app/src/components/pos/CartSidebar.tsx

```
import React from 'react';
import { CartItem } from '../../CartContext';

interface CartSidebarProps {
  items: CartItem[];
  onAddQty: (productId: string) => void;
  onSubQty: (productId: string) => void;
  onRemoveItem: (productId: string) => void;
}
const CartSidebar: React.FC<CartSidebarProps> = ({ items, onAddQty, onSubQty,
onRemoveItem }) => {
  const total = items.reduce((sum, it) => sum + it.price * it.quantity, 0);
  return (
    <div className="bg-white dark:bg-gray-800 rounded-lg shadow p-4 flex flex-
col max-h-[60vh]">
      <h2 className="text-xl font-bold mb-3 text-gray-800 dark:text-
gray-100">Cart</h2>
      <div className="flex-1 overflow-y-auto pr-1">
        {items.length === 0 ? (
          <p className="text-gray-500 dark:text-gray-400">Your cart is empty.</
p>
        ) : (
          <ul className="space-y-2">
            {items.map(item => {
              const isInactive = !item /* placeholder: we might mark inactive
items here if needed */;
              return (
                <li key={item.id} className="flex items-center justify-between
bg-gray-50 dark:bg-gray-700 rounded px-3 py-2">
                  <div className="flex-1 mr-2">
                    <div className="text-sm font-semibold text-gray-800
dark:text-gray-100">
                      {item.name}{/* Mark if inactive: */}{/* isInactive &&
<span className="text-red-600 text-xs ml-1">(inactive)</span> */}
                    </div>
```

```
                  <div className="text-xs text-gray-500 dark:text-gray-400">
                    {item.quantity} × ${item.price.toFixed(2)}
                  </div>
                </div>
                <div className="flex items-center">
                  <button
                    className="px-2 text-lg text-gray-700 dark:text-gray-200
disabled:opacity-50"
                    onClick={() => onSubQty(item.id)}
                    disabled={item.quantity <= 1}
                    aria-label="Decrease quantity"
                  >
                    &minus;
                  </button>
                  <span className="px-2 text-sm">{item.quantity}</span>
                  <button
                    className="px-2 text-lg text-gray-700 dark:text-gray-200"
                    onClick={() => onAddQty(item.id)}
                    aria-label="Increase quantity"
                  >
                    +
                  </button>
                  <button
                    className="ml-3 text-red-600 dark:text-red-400 text-xl"
                    onClick={() => onRemoveItem(item.id)}
                    title="Remove item"
                  >
                    &times;
                  </button>
                </div>
              </li>
            );
          })}
        </ul>
      )}
    </div>
    <div className="mt-3 pt-3 border-t border-gray-200 dark:border-gray-700
text-lg font-semibold flex justify-between text-gray-800 dark:text-gray-100">
      <span>Total:</span>
      <span>${total.toFixed(2)}</span>
    </div>
  </div>
 );
};
export default CartSidebar;
```

frontends/pos-app/src/components/pos/SubmitSalePanel.tsx

```
import React from 'react';
import type { PaymentMethod } from '../../OrderContext';

interface SubmitSalePanelProps {
  total: number;
  paymentMethod: PaymentMethod;
  onPaymentMethodChange: (method: PaymentMethod) => void;
  onSubmit: () => void;
  submitting: boolean;
}
const SubmitSalePanel: React.FC<SubmitSalePanelProps> = ({ total,
paymentMethod, onPaymentMethodChange, onSubmit, submitting }) => (
  <div className="mt-4 bg-white dark:bg-gray-800 rounded-lg shadow p-4">
    <div className="flex items-center justify-between mb-3">
      <span className="text-lg font-bold text-gray-800 dark:text-
gray-100">Total: ${total.toFixed(2)}</span>
      <select
        value={paymentMethod}
        onChange={e => onPaymentMethodChange(e.target.value as PaymentMethod)}
        className="px-3 py-1 border border-gray-300 rounded focus:outline-none
focus:ring-1 focus:ring-cyan-500"
        aria-label="Select payment method"
      >
        <option value="cash">Cash</option>
        <option value="card">Card</option>
        <option value="crypto">Crypto</option>
      </select>
    </div>
    <button
      onClick={onSubmit}
      disabled={submitting}
      className="w-full py-2 bg-cyan-600 text-white font-semibold rounded
hover:bg-cyan-700 disabled:opacity-50"
    >
      {submitting ? 'Processing...' : 'Submit Sale'}
    </button>
  </div>
);
export default SubmitSalePanel;
```

frontends/pos-app/src/components/pos/RecentOrdersDrawer.tsx

```
import React, { useEffect } from 'react';
import { useOrders } from '../../OrderContext';
```

```
interface RecentOrdersDrawerProps {
  open: boolean;
  onClose: () => void;
}
const RecentOrdersDrawer: React.FC<RecentOrdersDrawerProps> = ({ open,
onClose }) => {
  const { recentOrders } = useOrders();
  // Close on Escape key
  useEffect(() => {
    if (!open) return;
    const handleKey = (e: KeyboardEvent) => { if (e.key === 'Escape')
onClose(); };
    window.addEventListener('keydown', handleKey);
    return () => window.removeEventListener('keydown', handleKey);
  }, [open, onClose]);

  if (!open) return null;
  return (
    <div className="fixed inset-0 z-50 flex">
      {/* Overlay */}
      <div className="absolute inset-0 bg-black bg-opacity-50" onClick={onClose}
></div>
      {/* Drawer panel */}
      <div className="relative ml-auto h-full w-full max-w-sm bg-white dark:bg-
gray-800 shadow-xl p-4 overflow-y-auto">
        <h2 className="text-xl font-bold mb-4 text-gray-800 dark:text-
gray-100">Recent Orders</h2>
        {recentOrders.length === 0 ? (
          <p className="text-gray-600 dark:text-gray-400">No recent orders.</p>
        ) : (
          <ul className="space-y-3">
            {recentOrders.map(order => (
              <li key={order.reference || order.id} className="p-3 bg-gray-50
dark:bg-gray-700 rounded">
                <div className="flex justify-between text-sm">
                  <span className="font-semibold">Ref: {order.reference}</span>
                  <span>{new Date(order.createdAt).toLocaleString()}</span>
                </div>
                <div className="text-xs mt-1">
                  Status: <span className="font-medium">{order.status}</
span><br/>
                  {order.paymentStatus && <>Payment: <span className="font-
medium">{order.paymentStatus}</span><br/></>}
                  Total: ${order.total.toFixed(2)}<br/>
                  {order.offline && <span className="text-amber-600">Queued
(offline)</span>}
                  {order.note && <span className="text-red-600">Error:
```

```
{order.note}</span>}
                </div>
                {order.paymentUrl && (
                  <a href={order.paymentUrl} target="_blank" rel="noopener
noreferrer" className="text-cyan-700 text-xs underline mt-1 inline-block">
                    Complete Payment
                  </a>
                )}
              </li>
            ))}
          </ul>
        )}
        <button onClick={onClose} className="absolute top-2 right-2 text-2xl
text-gray-500 hover:text-gray-700">&times;</button>
      </div>
    </div>
  );
};
export default RecentOrdersDrawer;
```

frontends/pos-app/src/components/pos/ReplaceItemModal.tsx

```
import React, { useState } from 'react';
import { CartItem } from '../../CartContext';
import type { Product } from '../../hooks/useProducts';

interface ReplaceItemModalProps {
  item: CartItem;
  products: Product[];  // active products list
  onReplace: (replacementProductId: string) => void;
  onRemove: () => void;
  onCancel: () => void;
}
const ReplaceItemModal: React.FC<ReplaceItemModalProps> = ({ item, products,
onReplace, onRemove, onCancel }) => {
  const [replacementId, setReplacementId] = useState<string>('');
  return (
    <div className="fixed inset-0 z-50 flex items-center justify-center">
      {/* Overlay */}
      <div className="absolute inset-0 bg-black bg-opacity-50"
onClick={onCancel}></div>
      {/* Modal content */}
      <div className="relative bg-white dark:bg-gray-800 rounded-lg shadow-lg
p-6 w-11/12 max-w-sm">
        <h3 className="text-lg font-bold mb-4 text-gray-800 dark:text-
gray-100">Item Unavailable</h3>
```

```
        <p className="text-sm text-gray-700 dark:text-gray-300 mb-4">
          <strong>{item.name}</strong> is no longer available for sale. You can
replace it with another item or remove it from the cart.
        </p>
        <div className="mb-4">
          <label className="block text-sm font-medium mb-1">Replace with:</
label>
          <select
            className="w-full border border-gray-300 rounded px-3 py-2"
            value={replacementId}
            onChange={e => setReplacementId(e.target.value)}
          >
            <option value="">-- Select Product --</option>
            {products.filter(p => p.id !== item.id).map(product => (
              <option key={product.id} value={product.id}>{product.name} - $
{product.price.toFixed(2)}</option>
            ))}
          </select>
        </div>
        <div className="flex items-center justify-end gap-2">
          <button className="px-4 py-2 bg-gray-200 dark:bg-gray-700 rounded"
onClick={onCancel}>Cancel</button>
          <button
            className="px-4 py-2 bg-red-600 text-white rounded"
            onClick={onRemove}
          >
            Remove Item
          </button>
          <button
            className="px-4 py-2 bg-cyan-600 text-white rounded
disabled:opacity-50"
            disabled={!replacementId}
            onClick={() => onReplace(replacementId)}
          >
            Replace
          </button>
        </div>
      </div>
    </div>
  );
};
export default ReplaceItemModal;
```

Hooks

`frontends/pos-app/src/hooks/useProducts.ts`

```
import { useEffect, useState } from 'react';
import { useAuth } from '../AuthContext';

// Product type including fields from product-service
export interface Product {
  id: string;
  name: string;
  price: number;
  description: string;
  active: boolean;
  sku?: string | null;
  image_url?: string | null;
  category?: string;
}

interface UseProductsResult {
  products: Product[];
  categories: string[];
  isLoading: boolean;
  isOfflineResult: boolean;
  error: string | null;
  reload: () => void;
}
export const useProducts = (): UseProductsResult => {
  const { currentUser, token } = useAuth();
  const tenantId = currentUser?.tenant_id ? String(currentUser.tenant_id) :
null;
  const PRODUCT_SERVICE_URL = (import.meta.env.VITE_PRODUCT_SERVICE_URL ??
'http://localhost:8081').replace(/\/$/, '');

  const [products, setProducts] = useState<Product[]>([]);
  const [categories, setCategories] = useState<string[]>(['All']);
  const [isLoading, setIsLoading] = useState<boolean>(false);
  const [isOfflineResult, setIsOfflineResult] = useState<boolean>(false);
  const [error, setError] = useState<string | null>(null);

  // Helper to normalize raw API data to Product type
  const normalizeProducts = (data: any[]): Product[] => {
    const list: Product[] = [];
    for (const item of data) {
      if (!item || typeof item !== 'object') continue;
      const id = item.id;
```

```
      const name = item.name;
      const price = typeof item.price === 'number' ? item.price :
Number(item.price);
      if (typeof id !== 'string' && typeof id !== 'number') continue;
      if (typeof name !== 'string' || Number.isNaN(price)) continue;
      list.push({
        id: typeof id === 'string' ? id : String(id),
        name,
        price,
        description: item.description ?? '',
        active: item.active !== false,
        sku: item.sku ?? null,
        image_url: item.image_url ?? null,
      });
    }
    return list;
  };

  const loadProducts = async () => {
    if (!tenantId) {
      setProducts([]);
      return;
    }
    setIsLoading(true);
    setError(null);
    setIsOfflineResult(false);
    const headers: HeadersInit = { 'X-Tenant-ID': tenantId };
    if (token) headers['Authorization'] = `Bearer ${token}`;
    try {
      const res = await fetch(`${PRODUCT_SERVICE_URL}/products`, { headers });
      if (!res.ok) throw new Error(`Failed to fetch products (${res.status})`);
      const rawData = await res.json();
      let prodList = normalizeProducts(Array.isArray(rawData) ? rawData : []);
      // Cache the catalog in localStorage for offline use
      localStorage.setItem(`productCache:${tenantId}`,
JSON.stringify(prodList));
      // Filter only active products for display
      prodList = prodList.filter(p => p.active);
      setProducts(prodList);
      // Derive categories from product data (if available in description or
future field)
      const cats = new Set<string>(['All']);
      for (const p of prodList) {
        if (p.category && p.category.trim()) {
          cats.add(p.category.trim());
        } else if (p.description) {
          // If category not explicitly provided, attempt to parse from
description format "Category: X"
```

```
          const match = p.description.match(/Category:\s*([\w\s]+)/i);
          if (match) cats.add(match[1].trim());
        }
      }
      setCategories(Array.from(cats));
    } catch (err) {
      console.warn('Unable to load products', err);
      // Load from cache on failure (offline)
      try {
        const cached = localStorage.getItem(`productCache:${tenantId}`);
        if (cached) {
          const parsed = JSON.parse(cached);
          const prodList = normalizeProducts(Array.isArray(parsed) ? parsed :
[]);
          const activeList = prodList.filter(p => p.active);
          setProducts(activeList);
          setIsOfflineResult(true);
        } else {
          setProducts([]);
        }
      } catch {
        setProducts([]);
      }
      setError('Product catalog unavailable offline.');
    } finally {
      setIsLoading(false);
    }
  };

  useEffect(() => {
    loadProducts();
    // Optionally reload when tenant changes
  }, [tenantId, token]);

  return { products, categories, isLoading, isOfflineResult, error, reload:
loadProducts };
};
```

frontends/pos-app/src/hooks/useSubmitOrder.ts

```
import { useCallback, useRef, useState } from 'react';
import { useOrders } from '../OrderContext';
import type { DraftOrderPayload, SubmitOrderResult } from '../OrderContext';

interface UseSubmitOrderResult {
  submit: (order: DraftOrderPayload) => Promise<SubmitOrderResult | null>;
```

```
  submitting: boolean;
}
export const useSubmitOrder = (): UseSubmitOrderResult => {
  const { submitOrder } = useOrders();
  const [submitting, setSubmitting] = useState<boolean>(false);
  const lastSubmitTime = useRef<number>(0);

  const wrappedSubmit = useCallback(async (orderPayload: DraftOrderPayload) => {
    const now = Date.now();
    if (submitting || now - lastSubmitTime.current < 1000) {
      // Prevent rapid double submissions within 1 second
      return null;
    }
    setSubmitting(true);
    lastSubmitTime.current = now;
    try {
      const result = await submitOrder(orderPayload);
      return result;
    } catch (error) {
      throw error;
    } finally {
      setSubmitting(false);
    }
  }, [submitOrder, submitting]);

  return { submit: wrappedSubmit, submitting };
};
```

Note: The `DraftOrderPayload` and `SubmitOrderResult` types are assumed to be exported from `OrderContext.tsx` (they correspond to the order submission payload and result structure).

`frontends/pos-app/src/hooks/useOfflineQueue.ts`

```
import { useOrders } from '../OrderContext';

export const useOfflineQueue = () => {
  const { queuedOrders, isSyncing, retryQueue } = useOrders();
  return {
    queuedOrders,
    isSyncing,
    retryQueue: async () => {
      // Manually trigger retry of offline queue
      await retryQueue();
    },
  };
};
```

`frontends/pos-app/src/hooks/useSyncOnReconnect.ts`

```
import { useEffect } from 'react';
import { useOrders } from '../OrderContext';

/**
 * Hook to automatically sync queued orders when the browser reconnects online.
 */
export const useSyncOnReconnect = () => {
  const { retryQueue } = useOrders();
  useEffect(() => {
    const handleOnline = () => {
      retryQueue().catch(err => console.warn('Sync on reconnect failed', err));
    };
    window.addEventListener('online', handleOnline);
    return () => {
      window.removeEventListener('online', handleOnline);
    };
  }, [retryQueue]);
};
```

Tests

Below are unit tests for core logic (Cart and Order offline queue) and an integration test for key flows (adding items, filtering, submitting orders offline and online). These tests use **Vitest** and React Testing Library to simulate usage of the POS screen.

`frontends/pos-app/src/CartContext.test.tsx`

```
import { renderHook, act } from '@testing-library/react';
import { CartProvider, useCart } from './CartContext';

describe('CartContext', () => {
  it('adds items and updates total and quantity correctly', () => {
    const wrapper: React.FC<{ children: React.ReactNode }> = ({ children }) =>
<CartProvider>{children}</CartProvider>;
    const { result } = renderHook(() => useCart(), { wrapper });

    // Initial cart empty
    expect(result.current.cart).toEqual([]);
    expect(result.current.totalAmount).toBe(0);

    // Add a product
    act(() => {
      result.current.addItem({ id: 'prod1', name: 'Test Product', price: 10 });
```

```
    });
    expect(result.current.cart.length).toBe(1);
    expect(result.current.cart[0].quantity).toBe(1);
    expect(result.current.totalAmount).toBe(10);

    // Add same product again (should increment quantity)
    act(() => {
      result.current.addItem({ id: 'prod1', name: 'Test Product', price: 10 });
    });
    expect(result.current.cart.length).toBe(1);
    expect(result.current.cart[0].quantity).toBe(2);
    expect(result.current.totalAmount).toBe(20);

    // Increment quantity using incrementItemQuantity
    act(() => {
      result.current.incrementItemQuantity('prod1');
    });
    expect(result.current.cart[0].quantity).toBe(3);
    expect(result.current.totalAmount).toBe(30);

    // Decrement quantity
    act(() => {
      result.current.decrementItemQuantity('prod1');
    });
    expect(result.current.cart[0].quantity).toBe(2);

    // Remove item
    act(() => {
      result.current.removeItem('prod1');
    });
    expect(result.current.cart.length).toBe(0);
    expect(result.current.totalAmount).toBe(0);
  });
});
```

frontends/pos-app/src/OrderContext.test.tsx

```
import { renderHook, act } from '@testing-library/react';
import { AuthProvider } from './AuthContext';
import { OrderProvider, useOrders } from './OrderContext';

// Utility to wrap with Auth + Order providers
const wrapper: React.FC<{ children: React.ReactNode }> = ({ children }) => (
  <AuthProvider>
    <OrderProvider>{children}</OrderProvider>
  </AuthProvider>
```

```
);

describe('OrderContext offline queue', () => {
  beforeEach(() => {
    // Set up a valid session (tenant and token) for AuthContext
    const session = { token: 'test-token', user: { tenant_id: 'tenant-123' },
timestamp: Date.now() };
    window.localStorage.setItem('session', JSON.stringify(session));
    // Simulate offline
    Object.defineProperty(window.navigator, 'onLine', { writable: true, value:
false });
    // Clear any stored offline orders
    window.localStorage.removeItem('pos-offline-orders');
    window.localStorage.removeItem('pos-recent-orders');
  });
  afterEach(() => {
    // Clean up navigator onLine
    Object.defineProperty(window.navigator, 'onLine', { writable: true, value:
true });
    window.localStorage.clear();
  });
  it('queues orders when offline and flushes on reconnect', async () => {
    const { result } = renderHook(() => useOrders(), { wrapper });
    // Prepare a draft order payload
    const draftOrder = {
      items: [{ product_id: 'p1', quantity: 1, unit_price: 5.0, line_total:
5.0 }],
      payment_method: 'cash' as const,
      total: 5.0,
    };
    // Submit order while offline (should queue)
    let submitResult;
    await act(async () => {
      submitResult = await result.current.submitOrder(draftOrder);
    });
    expect(submitResult.status).toBe('queued');
    expect(result.current.queuedOrders.length).toBe(1);
    // The recentOrders should include a queued offline entry
    expect(result.current.recentOrders[0].offline).toBe(true);
    expect(result.current.recentOrders[0].status).toMatch(/Queued/);

    // Simulate coming online and flushing queue
    Object.defineProperty(window.navigator, 'onLine', { writable: true, value:
true });
    await act(async () => {
      await result.current.retryQueue();
    });
    // After sync, queuedOrders should be empty
```

```
    expect(result.current.queuedOrders.length).toBe(0);
    // Recent orders entry should be updated to non-offline with an actual id
and possibly status
    const updatedOrder = result.current.recentOrders.find(o => o.reference && !
o.offline);
    expect(updatedOrder).toBeDefined();
    expect(updatedOrder.status.toLowerCase()).not.toContain('queued');
  });
});
```

frontends/pos-app/src/pages/CashierPage.test.tsx

```
import React from 'react';
import { describe, it, expect, beforeEach, vi } from 'vitest';
import { render, fireEvent, waitFor, screen } from '@testing-library/react';
import { AuthProvider } from '../AuthContext';
import { OrderProvider } from '../OrderContext';
import { CartProvider } from '../CartContext';
import CashierPage from './CashierPage';

// Helper to render CashierPage with all providers
const renderWithProviders = () => {
  return render(
    <AuthProvider>
      <OrderProvider>
        <CartProvider>
          <CashierPage />
        </CartProvider>
      </OrderProvider>
    </AuthProvider>
  );
};

describe('CashierPage integration flows', () => {
  beforeEach(() => {
    // Set up a logged-in session for AuthContext
    const session = { token: 'token-123', user: { tenant_id: 'tenant-test' },
timestamp: Date.now() };
    localStorage.setItem('session', JSON.stringify(session));
    // Mock fetch for products and order submissions
    vi.stubGlobal('fetch', vi.fn((input: RequestInfo, init?: RequestInit) => {
      const url = typeof input === 'string' ? input : input.url;
      // Return product list for product service
      if (url.includes('/products')) {
        const responseBody = [
          { id: 'p1', name: 'Product One', price: 10, description: '', active:
```

```
true },
          { id: 'p2', name: 'Product Two', price: 20, description: '', active:
true },
          // Include an inactive product to test replacement flow
          { id: 'pX', name: 'Old Product', price: 15, description: '', active:
false },
        ];
        return Promise.resolve(new Response(JSON.stringify(responseBody), {
status: 200 }));
      }
      // Handle order submission
      if (url.includes('/orders')) {
        const orderResponse = { id: 'order-123', status: 'Submitted' };
        return Promise.resolve(new Response(JSON.stringify(orderResponse), {
status: 200 }));
      }
      if (url.includes('/payments')) {
        const payResponse = { status: 'pending', payment_url: 'http://pay.test/
abc' };
        return Promise.resolve(new Response(JSON.stringify(payResponse), {
status: 200 }));
      }
      return Promise.resolve(new Response(null, { status: 500 }));
    }));
  });

  afterEach(() => {
    vi.restoreAllMocks();
    localStorage.clear();
  });

  it('loads products and allows adding to cart and filtering', async () => {
    renderWithProviders();
    // Wait for product list to load
    await screen.findByText('Product One');
    expect(screen.getByText('Product Two')).toBeInTheDocument();

    // Search filtering
    const searchInput = screen.getByPlaceholderText('Search products...');
    fireEvent.change(searchInput, { target: { value: 'Product One' } });
    expect(screen.queryByText('Product Two')).not.toBeInTheDocument();
    expect(screen.getByText('Product One')).toBeInTheDocument();
  });

  it('adds items to cart and updates total', async () => {
    renderWithProviders();
    await screen.findByText('Product One');
    // Click "Add" on Product One twice
```

```
    const addButtons = screen.getAllByText('Add');
    fireEvent.click(addButtons[0]);
    fireEvent.click(addButtons[0]);
    // Cart should show Product One with quantity 2 and correct total
    await screen.findByText('Test Product');
    const qtyElem = screen.getByText('2');
    expect(qtyElem).toBeInTheDocument();
    expect(screen.getByText('$20.00')).toBeInTheDocument();  // total for 2 ×
$10
  });

  it('submits an order offline (queues) then online (direct submission)', async
() => {
    // Initialize with a cart containing an inactive item (pX) from localStorage
to test replace flow
    const offlineCart = [{ id: 'pX', name: 'Old Product', price: 15, quantity:
1 }];
    localStorage.setItem('pos-cart', JSON.stringify(offlineCart));
    // Render while offline
    Object.defineProperty(navigator, 'onLine', { value: false, configurable:
true });
    renderWithProviders();
    // Should detect offline mode, show offline banner
    expect(await screen.findByText(/Offline mode/)).toBeInTheDocument();
    // Submit order (offline)
    const submitBtn = await screen.findByText('Submit Sale');
    fireEvent.click(submitBtn);
    // Expect the ReplaceItemModal to appear for the inactive item
    await screen.findByText('Item Unavailable');
    // Choose replacement product and replace
    fireEvent.change(screen.getByLabelText('Replace with:'), { target: { value:
'p1' } });
    fireEvent.click(screen.getByText('Replace'));
    // After replacing, submission should proceed (queue offline since still
offline)
    const ordersButton = screen.getByText(/Orders/);
    expect(ordersButton.textContent).toMatch(/\(1\)$/);  // queued count
increments
    // Now simulate going online and flush queue
    Object.defineProperty(navigator, 'onLine', { value: true, configurable:
true });
    // Trigger sync (e.g., via the Retry Sync button)
    fireEvent.click(screen.getByText('Retry Sync'));
    await waitFor(() => {
      // After sync, Orders button count should reset to 0
      expect(screen.getByText('Orders').textContent).not.toContain('(');
    });
    // Now test direct online submission flow:
```

```
    // Add a product to cart and submit while online
    fireEvent.click(screen.getAllByText('Add')[0]);  // add Product One
    Object.defineProperty(navigator, 'onLine', { value: true, configurable:
true });
    fireEvent.click(screen.getByText('Submit Sale'));
    // Should clear cart and add recent order entry (check drawer for submitted
order)
    fireEvent.click(screen.getByText(/Orders/));
    const recentOrder = await screen.findByText(/Ref:/);
    expect(recentOrder).toBeInTheDocument();
  });
});
```

Notes on testing: We use a stubbed `fetch` to simulate API responses. The integration test covers: product loading and search filtering, adding products to cart, offline submission with item replacement, and online submission with clearing of the cart and recent orders listing. Adjust the test data or assertions as needed to match actual API responses and UI text.
