User Dashboard Implementation Plan

The Petal Pouches E-Commerce Platform

Project: Customer-facing dashboard mirroring admin functionality
Based on: Admin dashboard with TPP theme and component system
Database: Supabase PostgreSQL (25 tables, UUID + JSONB)
Frontend Stack: React 18, Tailwind CSS, Lucide Icons

---

1. DATABASE SCHEMA ANALYSIS

Core Customer Tables

Table	Purpose	Key Fields
users	Customer accounts	id, name, email, password_hash, created_at
addresses	Delivery addresses	id, user_id, line1-2, city, state, country, zip_code
carts	Shopping carts	id, user_id, session_id, expires_at
cart_items	Cart products	id, cart_id, product_variant_id, quantity
orders	Order records	id, user_id, status, final_total, payment_status
order_items	Order line items	id, order_id, product_variant_id, price, quantity
payments	Payment records	id, order_id, provider (Razorpay/Stripe), status
wishlist	Saved items	id, user_id, product_id
reviews	Product reviews	id, product_id, user_id, rating (1-5), comment
coupons	Discount codes	code, discount_type, discount_value, usage_limit
gift_previews	Personalization	id, user_id, personalization (JSONB)
bundles_custom	User bundles	id, user_id, items, total_price, is_published
bundle_user_public	Shared bundles	id, custom_bundle_id, upvote_count

Key Relationships

- User → Addresses (1:Many)
- User → Orders (1:Many)
- User → Cart (1:1)
- Order → OrderItems → ProductVariants (Many:Many)
- User → Reviews (1:Many)

- User → Wishlist (1:Many)

---

2. DIRECTORY STRUCTURE

```
frontend/src/
├── pages/customer/
│   ├── Dashboard.jsx          # Home page
│   ├── Profile.jsx            # Account settings
│   ├── Addresses.jsx          # Manage addresses
│   ├── Orders.jsx             # Order history
│   ├── OrderDetails.jsx       # Single order view
│   ├── Wishlist.jsx           # Saved items
│   ├── Cart.jsx               # Shopping cart
│   ├── Checkout.jsx           # Checkout flow
│   └── Settings.jsx           # Security settings
│
├── components/customer/
│   ├── layout/
│   │   ├── CustomerLayout.jsx
│   │   ├── CustomerSidebar.jsx
│   │   └── CustomerHeader.jsx
│   │
│   ├── profile/
│   │   ├── ProfileForm.jsx
│   │   ├── PasswordChange.jsx
│   │   └── AvatarUpload.jsx
│   │
│   ├── addresses/
│   │   ├── AddressCard.jsx
│   │   ├── AddressForm.jsx
│   │   ├── AddressList.jsx
│   │   ├── AddressAutocomplete.jsx
│   │   └── AddressMapPicker.jsx
│   │
│   ├── orders/
│   │   ├── OrderCard.jsx
│   │   ├── OrderStatusTimeline.jsx
│   │   ├── OrderItems.jsx
│   │   └── TrackingInfo.jsx
│   │
│   ├── cart/
│   │   ├── CartItem.jsx
```

```
│   │   ├── CartSummary.jsx
│   │   ├── CouponInput.jsx
│   │   ├── PromoCode.jsx
│   │   └── EmptyCart.jsx
│   │
│   ├── wishlist/
│   │   ├── WishlistCard.jsx
│   │   └── WishlistFilter.jsx
│   │
│   ├── checkout/
│   │   ├── ShippingStep.jsx
│   │   ├── PaymentStep.jsx
│   │   ├── PersonalizationStep.jsx
│   │   └── ReviewStep.jsx
│   │
│   └── reviews/
│       ├── ReviewForm.jsx
│       ├── ReviewCard.jsx
│       ├── RatingStars.jsx
│       └── ReviewList.jsx
│
├── services/
│   ├── userAuthService.js      # Login/Register/Auth
│   ├── orderService.js         # Order CRUD
│   ├── addressService.js       # Address CRUD
│   ├── cartService.js          # Cart operations
│   ├── wishlistService.js      # Wishlist operations
│   ├── reviewService.js        # Review CRUD
│   ├── paymentService.js       # Razorpay/Stripe integration
│   ├── geocodingService.js     # Mapbox + Nominatim
│   └── personalizeService.js   # Gift personalization
│
├── context/
│   ├── UserAuthContext.jsx     # Auth state
│   ├── CartContext.jsx         # Cart state
│   └── WishlistContext.jsx     # Wishlist state
│
├── hooks/
│   ├── useCustomerAuth.js      # Auth logic
│   ├── useCart.js              # Cart operations
│   └── useAddress.js           # Address validation
│
└── utils/
    ├── customerHelpers.js      # Address formatting, validation
```

```
├── paymentHelpers.js       # Payment utilities
└── geocodingHelpers.js     # Map utilities
```

---

3. BACKEND ENDPOINTS (Express.js)

Authentication

```
POST   /api/auth/register            # Email/password registration
POST   /api/auth/login               # Login
POST   /api/auth/logout              # Logout
POST   /api/auth/refresh             # Token refresh
POST   /api/auth/verify-email        # Email verification
POST   /api/auth/forgot-password     # Password reset
POST   /api/auth/reset-password/:token # Reset token
GET    /api/auth/me                  # Current user profile
```

User Profile

```
GET    /api/users/profile            # Get profile
PUT    /api/users/profile            # Update profile
POST   /api/users/avatar             # Upload avatar (Cloudinary)
PUT    /api/users/password           # Change password
DELETE /api/users/account            # Delete account
```

Addresses

```
GET    /api/addresses                # List user addresses
POST   /api/addresses                # Create address
PUT    /api/addresses/:id            # Update address
DELETE /api/addresses/:id            # Delete address
PATCH  /api/addresses/:id/default    # Set default
POST   /api/addresses/geocode        # Nominatim/Mapbox search
```

Shopping Cart

```
GET    /api/cart                     # Get user cart
POST   /api/cart/items               # Add to cart
PATCH  /api/cart/items/:id           # Update quantity
DELETE /api/cart/items/:id           # Remove from cart
```

```
DELETE /api/cart                  # Clear cart
POST   /api/cart/apply-coupon     # Apply discount code
```

Orders

```
GET    /api/orders                # List user orders
GET    /api/orders/:id            # Order details
POST   /api/orders                # Create order
PATCH  /api/orders/:id/status     # Update status (admin only)
GET    /api/orders/:id/tracking   # Tracking info
POST   /api/orders/:id/cancel     # Cancel order
```

Payments (Razorpay/Stripe)

```
POST   /api/payments/razorpay/create   # Create Razorpay order
POST   /api/payments/razorpay/verify   # Verify payment
POST   /api/payments/stripe/intent     # Create payment intent
POST   /api/payments/stripe/webhook    # Stripe webhook
```

Reviews & Wishlist

```
POST   /api/reviews               # Create review
GET    /api/reviews/product/:id   # Get product reviews
DELETE /api/reviews/:id           # Delete review
GET    /api/wishlist              # List wishlist
POST   /api/wishlist              # Add to wishlist
DELETE /api/wishlist/:id          # Remove from wishlist
```

---

4. FILE SPECIFICATIONS

4.1 Authentication Layer

userAuthService.js (60 lines)

- `register(email, password, name)` - Create account, hash password, store in users table
- `login(email, password)` - Verify credentials, generate JWT token
- `logout()` - Clear token from localStorage
- `getCurrentUser()` - Fetch /api/auth/me with token
- `refreshToken()` - Renew expiring tokens

- `requestPasswordReset(email)` - Send reset email
- `resetPassword(token, newPassword)` - Verify token and update

useCustomerAuth.js (80 lines)

- Hook for authentication state
- `isAuthenticated`, `user`, `loading`, `error`
- Persist token to localStorage
- Auto-refresh token on app load
- Logout and clear state

4.2 Address Management

AddressForm.jsx (150 lines)

- Form component for adding/editing addresses
- Fields: line1, line2, city, state, country, zip_code, landmark, phone
- Uses AddressAutocomplete for search suggestions
- Uses AddressMapPicker for map selection

AddressAutocomplete.jsx (120 lines)

- **Nominatim Integration**: Fetch suggestions from OpenStreetMap
- Query: `/nominatim/search.php?q=${query}&format=json&addressdetails=1`
- Debounced input (500ms)
- Display 5 results dropdown
- Return { address, lat, lng }

AddressMapPicker.jsx (200 lines)

- **Mapbox + Leaflet.js combo**:
  - Mapbox as primary (better UX, 50k requests/month free)
  - Fallback to OpenStreetMap with Leaflet
- Embedded map with marker drag
- Click-to-select location
- Reverse geocoding to get full address

addressService.js (60 lines)

- `getAddresses()` - Fetch from /api/addresses
- `createAddress(data)` - POST to backend
- `updateAddress(id, data)` - PUT request
- `deleteAddress(id)` - DELETE request
- `setDefault(id)` - PATCH /api/addresses/:id/default

4.3 Cart System

cartService.js (80 lines)

- `getCart()` - Fetch user cart with items
- `addToCart(variantId, quantity)` - POST /api/cart/items
- `updateQuantity(itemId, quantity)` - PATCH
- `removeItem(itemId)` - DELETE
- `clearCart()` - DELETE /api/cart
- `applyCoupon(code)` - POST /api/cart/apply-coupon

CartItem.jsx (100 lines)

- Display product image, title, variant details
- Quantity selector with +/- buttons
- Remove button
- Show unit price and item total
- Show discount if applicable

CartSummary.jsx (80 lines)

- Subtotal calculation
- Discount display
- Shipping cost
- Tax calculation
- Final total
- Checkout button (disabled if cart empty)

CouponInput.jsx (60 lines)

- Input for coupon code
- Apply button
- Validation messages
- Show applied coupon and discount amount

4.4 Orders

orderService.js (70 lines)

- `getOrders(filters)` - GET /api/orders with pagination
- `getOrderById(id)` - Fetch single order with items
- `createOrder(cartItems, address, payment)` - POST /api/orders
- `cancelOrder(id)` - PATCH status to cancelled
- `getTracking(id)` - GET tracking info from shipment table

OrderCard.jsx (120 lines)

- Display order ID, date, status badge
- Total amount in TPP pink
- Item count
- Click to expand details
- Show action buttons (Track, Reorder, Return)

OrderStatusTimeline.jsx (100 lines)

- Visual timeline: Pending → Processing → Shipped → Delivered
- Current step highlighted
- Estimated dates
- Tracking number if shipped
- Show delivery date when completed

OrderDetails.jsx (150 lines)

- Full order information
- Order items with images and prices

- Shipping address
- Payment method
- Status timeline
- Download invoice button
- Review products button

4.5 Checkout & Payment

checkoutService.js (100 lines)

- `createRazorpayOrder(amount)` - Create order on Razorpay
- `verifyRazorpayPayment(response)` - Verify payment signature
- `createOrder(cartData, address, payment)` - Create order in DB
- `getCheckoutSummary(cart, address)` - Calculate totals

Checkout.jsx (300 lines)

- Step 1: Review cart
- Step 2: Select/add shipping address
- Step 3: Gift personalization (optional, JSONB)
- Step 4: Payment method (Razorpay/Stripe)
- Step 5: Review & confirm
- Progress indicator
- Persist progress to session storage

PaymentStep.jsx (120 lines)

- Radio buttons: Razorpay, Stripe, COD
- Razorpay:

```javascript
```

```
const options = {
  key: process.env.RAZORPAY_KEY_ID,
  amount: total * 100, // in paise
  currency: 'INR',
  name: 'The Petal Pouches',
  handler: verifyPayment,
  prefill: { email, contact }
};
```

- Stripe: loadStripe, useElements, useCardElement
- COD: Simple confirmation

PersonalizationStep.jsx (100 lines)

- Gift wrap option (yes/no)
- Gift message (max 500 chars)
- Recipient name & phone
- Store as JSONB in orders.gift_previews
- Preview text overlaid on product image

4.6 Reviews

ReviewForm.jsx (120 lines)

- Star rating selector (1-5)
- Comment textarea (max 1000 chars)
- Upload review images (Cloudinary, max 3)
- Post review to /api/reviews
- Show success/error messages

ReviewCard.jsx (80 lines)

- Customer name, avatar, date
- Star rating display
- Comment text (truncate if long)
- Review images carousel
- Helpful/Unhelpful buttons

RatingStars.jsx (40 lines)

- Reusable star display
- Editable mode: hover to select
- Display mode: show filled stars

4.7 Wishlist

WishlistCard.jsx (100 lines)

- Product image, title, price
- Stock status indicator
- Add to cart button
- Remove from wishlist button
- Show if in stock vs out of stock styling

wishlistService.js (50 lines)

- `getWishlist()` - GET /api/wishlist
- `addToWishlist(productId)` - POST
- `removeFromWishlist(id)` - DELETE
- `addAllToCart()` - Bulk operation

4.8 Layouts & Navigation

CustomerLayout.jsx (80 lines)

- Wrapper with header, sidebar, main content
- Sidebar: Dashboard, Profile, Addresses, Orders, Wishlist, Settings
- Header: User menu, Notifications, Cart icon
- Mobile: Hamburger menu

CustomerSidebar.jsx (120 lines)

- Navigation links with active state
- User profile section (avatar, name, email)
- Logout button
- Settings icon

- Collapsible on mobile

ProtectedCustomerRoute.jsx (60 lines)

- Check if user is authenticated
- Redirect to login if not
- Show loading spinner while checking
- Pass user data to children

4.9 Geocoding Utilities

geocodingService.js (180 lines)

```javascript
```

```
// Mapbox (Primary)
const mapboxSearch = async (query) => {
  const response = await fetch(
    `https://api.mapbox.com/geocoding/v5/mapbox.places/${query}.json?access_token=${MAPBOX_TOKEN}`
  );
  const data = await response.json();
  return data.features.map(f => ({
    address: f.place_name,
    lat: f.geometry.coordinates[1],
    lng: f.geometry.coordinates[0]
  }));
};

// Nominatim (Fallback)
const nominatimSearch = async (query) => {
  const response = await fetch(
    `https://nominatim.openstreetmap.org/search?q=${query}&format=json&addressdetails=1`
  );
  const data = await response.json();
  return data.map(r => ({
    address: r.display_name,
    lat: parseFloat(r.lat),
    lng: parseFloat(r.lon)
  }));
};

// Reverse geocoding
const reverseGeocode = async (lat, lng) => {
  // Try Mapbox first, fallback to Nominatim
};
```

geocodingHelpers.js (100 lines)

- `formatAddress(addressObj)` - Convert DB to display format
- `validateAddressComplete(addressObj)` - Check required fields
- `getDistanceKm(lat1, lng1, lat2, lng2)` - Haversine formula
- `parseAddressComponents(mapResponse)` - Extract city, state, country
- `buildAddressString(line1, line2, city, state, country, zip)` - Format for shipping

4.10 Context & Hooks

UserAuthContext.jsx (100 lines)

- State: user, isAuthenticated, loading, error
- Methods: login, register, logout, updateProfile
- Persist user to localStorage
- Auto-login on app load

CartContext.jsx (150 lines)

- State: items, cart, subtotal, tax, total, coupon
- Methods: addItem, removeItem, updateQuantity, applyCoupon, clearCart
- Sync with API on every change
- Calculate totals automatically

useCart.js (80 lines)

- `useCart()` hook
- Returns: items, addToCart, removeFromCart, total, etc.
- Wraps CartContext for easier access

---

5. GEO-MAPPING INTEGRATION

Option 1: Mapbox (RECOMMENDED)

- **Tier**: Free (50,000 monthly requests)
- **Setup**: Get API key from mapbox.com
- **CDN**: `https://api.mapbox.com/...`
- **Features**: Autocomplete, reverse geocoding, interactive maps
- **React**: Use `@react-map-gl/core` or `react-map-gl`

Option 2: OpenStreetMap + Nominatim + Leaflet.js

- **Tier**: Free (1 req/sec rate limit)
- **Setup**: No API key needed
- **CDN**: `https://nominatim.openstreetmap.org/...`

- **Maps**: Leaflet.js with OpenStreetMap tiles
- **React**: Use `react-leaflet`

Implementation Strategy

```javascript
// AddressAutocomplete.jsx
const [suggestions, setSuggestions] = useState([]);
const searchAddress = debounce(async (query) => {
  try {
    const results = await geocodingService.search(query);
    setSuggestions(results); // Show dropdown
  } catch (err) {
    console.error('Geocoding failed:', err);
  }
}, 500);
```

---

6. DATABASE OPERATIONS

User Registration

```sql
INSERT INTO users (id, name, email, password_hash, created_at)
VALUES (gen_random_uuid(), $1, $2, bcrypt($3), now())
```

Add to Cart

```sql
INSERT INTO cart_items (id, cart_id, product_variant_id, quantity)
VALUES (gen_random_uuid(), $1, $2, $3)
ON CONFLICT (cart_id, product_variant_id)
DO UPDATE SET quantity = quantity + $3
```

Create Order

```sql
```

```
BEGIN;
  INSERT INTO orders (id, user_id, subtotal, final_total, status, payment_status)
  VALUES ($1, $2, $3, $4, 'pending', 'unpaid');

  INSERT INTO order_items SELECT gen_random_uuid(), $1, product_variant_id, quantity, price
  FROM cart_items WHERE cart_id = $5;

  DELETE FROM cart_items WHERE cart_id = $5;
COMMIT;
```

---

7. AUTHENTICATION FLOW

1. User Register → Hash password → Insert into users table
2. User Login → Compare password → Generate JWT token
3. Store token in localStorage
4. Include token in Authorization header: "Bearer {token}"
5. On 401 response → Refresh token or redirect to login
6. Logout → Delete token from localStorage

---

8. IMPLEMENTATION PRIORITY

Phase 1 (Week 1-2) - Authentication & Core Layout

- User registration/login
- Protected routes
- CustomerLayout, Sidebar, Header
- Basic dashboard home page

Phase 2 (Week 2-3) - Address Management

- Add/edit addresses
- Mapbox + Nominatim integration
- Set default address

Phase 3 (Week 3) - Cart & Checkout

- Add to cart flow

- Cart page & summary
- Basic checkout (non-payment)

Phase 4 (Week 4) - Payments & Orders

- Razorpay integration
- Order creation & confirmation
- Order history view

Phase 5 (Week 5) - Secondary Features

- Reviews & ratings
- Wishlist
- Profile settings
- Order tracking

---

9. ENVIRONMENT VARIABLES

```env
# Frontend
VITE_API_BASE_URL=http://localhost:5000
VITE_MAPBOX_TOKEN=pk_live_xxx
VITE_STRIPE_KEY=pk_live_xxx

# Backend
RAZORPAY_KEY_ID=xxx
RAZORPAY_KEY_SECRET=xxx
JWT_SECRET=your-secret-key
```

---

10. TESTING CHECKLIST

- [ ] User registration with email verification
- [ ] Login/logout functionality
- [ ] Cart add/remove/update operations
- [ ] Address geocoding with both Mapbox & Nominatim
- [ ] Razorpay payment flow
- [ ] Order creation and status updates

- [ ] Wishlist operations
- [ ] Profile update
- [ ] Review submission
- [ ] Mobile responsiveness

Document Version: 1.0
Last Updated: November 2025
Theme: TPP Pink (#EC4899), Slate (#1E293B), Mint (#00D9A3)