E-COM Website: Technical Architecture (v2)

1. Overview

This document outlines the architecture for a simple, responsive e-commerce SPA. Built as a single HTML file, it simulates a multi-page experience by dynamically showing and hiding content sections.

Core functionalities include:

- A product catalog
- A persistent shopping cart
- A mock "My Orders" panel
- A multi-step checkout flow (cart, payment, confirmation)
- A customer review submission form

2. Tech Stack

- **Frontend (UI):** HTML & Tailwind CSS
- **Business Logic & State:** Vanilla JavaScript
- **Data (Backend Simulation):** Hardcoded JavaScript `const` arrays
- **Assets:** Inline Base64-encoded SVGs for reliable, dependency-free graphics

3. Rationale

This stack was chosen for its simplicity and zero-setup deployment (runs in any browser). Inline SVGs guarantee asset reliability, and `const` arrays simulate a lightweight backend.

JavaScript manages all state, which is split into:

1. **Data State:** A `cart` array, persisted in `localStorage`.
2. **UI State:** Helper functions (`showMainStore()` , etc.) toggle the visibility of `<div>` sections to simulate page navigation.

4. Architecture Breakdown

1. HTML Structure

The `<body>` contains a persistent `<header>` and several primary content `<div>` wrappers: `#main-content` (products, reviews, footer), `#payment-page` , and `#confirmation-page` . UI state functions hide or show these wrappers as needed. Additional modal `<div>` elements (`#cart-modal` , `#orders-modal` , `#loading-overlay`) are used for slide-out panels and overlays.

2. JavaScript Logic (`<script>`)

- **Data:** `const PRODUCTS` and `const ORDERS` arrays define all site data.
- **State:** `let cart = []` holds the live shopping cart.

- **Initialization (On Page Load):** `loadCartFromStorage()` runs first, followed by functions to render products (`renderProducts()`), the cart (`renderCart()`), orders (`renderOrders()`), and the review form dropdown (`populateReviewProducts()`) to build the initial UI.
- **Core Functions:**
  - **Rendering:** A set of `render...()` functions (e.g., `renderProducts()` , `renderCart()`) loop over the data/state arrays to generate and inject HTML.
  - **State & Actions:** Functions like `addToCart()` , `updateCartQuantity()` , and `removeFromCart()` modify the `cart` array. `saveCartToStorage()` persists these changes to `localStorage` .
  - **UI Management:** `toggle...Modal()` functions handle the slide-out panels. `showMainStore()` , `showPaymentPage()` , and `showConfirmationPage()` manage the main "page" visibility.
- **Event Listeners:**
  - Event delegation is used on `#product-grid` and `#cart-items` to manage clicks on dynamic content.
  - Listeners on header buttons and forms trigger the modal, review, and checkout functions.
  - **Checkout Flow:** A series of click handlers guides the user from `#checkout-button -> showPaymentPage() -> #pay-now-button -> showConfirmationPage() -> showMainStore()` , clearing the cart after the simulated payment.

5. Flowcharts

High-Level User Journey

```
graph TD
    A(Start: User visits site) --> B[View Product Catalog];
    B --> C(Click 'Add to Cart');
    C --> D(View Cart);
    D --> B;
    D --> E(Click 'Proceed to Checkout');
    E --> F[Fill Payment Details];
    F --> G(Click 'Pay Now');
    G --> H(Show 'Order Successful');
    H --> B;
    F --> B(Click 'Go Back');
    B --> I(Click 'My Orders');
    I --> B;
    B --> J[Write a Review];
    J --> B;
```

"Add to Cart" Application Logic

```
graph TD
    A(User clicks 'Add to Cart' button) --> B[Event listener captures click];
    B --> C{Get product ID};
    C --> D{Check 'cart' array for item?};
    D -- Yes --> E[Increment quantity];
    D -- No --> F[Add new object to 'cart' array];
```

```
E --> G[[renderCart()]];
F --> G;
G --> H(Update Cart UI);
G --> I(Update Header Count);
G --> J[[saveCartToStorage()]];
J --> K([localStorage]);
G --> L[[showAddedToCartMessage()]];
L --> M(Show 'Added!' on button);
M --> N(Open cart modal);
```