Darksword Armory Clone — Conversation Summary

Author: Manus AI
Date: February 14, 2026
Sessions: 2 (context carried over via task inheritance)

Project Timeline

This document summarizes every major decision, implementation step, and bug fix made during the development of the Darksword Armory e-commerce clone. It is intended to give the next agent full context on what was built, why certain choices were made, and what remains to be done.

Session 1: Initial Build (Feb 13–14, 2026)

Phase 1 — Design Template Analysis

The project began with an analysis of davidprotein.com as the design template. The client requested that the Darksword Armory website match davidprotein.com's design language exactly — including colors, dimensions, iframes, CTAs, headers, and tabs — while using Darksword Armory's product content and branding.

Key design decisions extracted from davidprotein.com included a dark background (#0A0A0A) with cream/gold text, Cormorant Garamond serif headings paired with DM Sans body text, a sticky header with transparent-to-solid scroll transition, a video hero section with auto-transition to image carousel, scrolling marquee trust bars, and a "peel-reveal" leather sheath effect on product cards.

The color palette was established as: background #0A0A0A, foreground #F5F0E8, gold accent #C8A45C, card background #111111, muted #1A1A1A, and border #2A2A2A.

Phase 2 — Homepage Construction

The homepage was built section by section to match the davidprotein.com layout. Each component was created as a standalone React component: AnnouncementBar (rotating messages with arrow navigation), Header (sticky with navigation, search, wishlist, account, cart icons), HeroSection (video background with auto-transition to image carousel after video ends, skip button, sound toggle), MarqueeBar (scrolling trust indicators), ProductCarousel (featured products with the peel-reveal effect), CraftsmanSection (founder Eyal Azerad profile), SpecsSection (tabbed specifications), BrandStory (brand narrative with parallax image), BundleSection (promotional bundle offer), CategoryShowcase (6-category grid with hover effects), ReviewsSection (customer testimonials carousel), and Footer (newsletter signup, navigation links, social media).

Hero images were generated using AI image generation to depict a medieval blacksmith forging a sword in a dark workshop, matching the brand's aesthetic. These images were uploaded to S3 CDN to avoid local file storage issues during deployment.

Phase 3 — Product Data Import

The client provided 31 WooCommerce XML export files containing the complete product catalog. A data extraction pipeline was built to parse these XML files and output structured JSON. The extraction yielded 367 products across 62 categories, with full metadata including titles, slugs, descriptions, excerpts, prices, SKUs, stock statuses, thumbnail URLs, gallery image URLs, and category assignments.

A critical discovery during this phase was that WooCommerce "variable" products store their prices in variation records, not in the main product record. This meant that 8 products initially showed $0.00 prices because the extraction script was reading the parent product price (which WooCommerce leaves empty for variable products) instead of computing the price range from variations.

Phase 4 — Full-Stack Upgrade

The project was upgraded from a static frontend to a full-stack application using the Manus `web-db-user` feature, which added Express, tRPC, Drizzle ORM, and Manus OAuth. The database schema was designed with 12 tables covering products, categories, product variations, product attributes, carts, cart items, orders, order items, users, addresses, wishlist items, and reviews.

The seed script (`seed-db.mjs`) was written to import all 367 products and 62 categories from the extracted JSON data. Default product variations (Package options and Blade Finish options) were initially generated programmatically for all sword products.

Phase 5 — Variation Data Re-Import

After the initial seed, the variation data was found to be incomplete — the default variations did not match the actual WooCommerce variation data. A second extraction pass was performed to pull all 1,150 variations directly from the XML files, preserving the exact option names, values, and prices from the original site.

The variation types discovered in the data were: Package (most common, e.g., “Blunt blade & Scabbard” at $580), Size, Model, Color, and Options. Each variation has an absolute price rather than a price modifier, which required updating the cart logic to use the variation’s price directly instead of adding a modifier to the base product price.

Phase 6 — Product Attributes

A separate extraction was performed for WooCommerce product attributes (taxonomy terms). These are non-price-affecting customization options: Grip (6 color swatches with hex values), Scabbard (1 color swatch), Blade Finish (dropdown with 2 options), and Guard & Pommel Finish (dropdown with 2 options). A new `product_attributes` table was created with a `display_type` enum (color_swatch/dropdown) and a `jsonb` options array. The UI was built with circular color swatch buttons for Grip/Scabbard and standard dropdown selectors for Blade Finish and Guard & Pommel Finish, matching the original darkswordarmory.com product page layout.

Phase 7 — Product Card Rewrite

The product cards were rewritten to match the original Darksword Armory site’s Flatsome WooCommerce theme. Key features implemented: product info displayed below the image (not overlaid), image hover swap (second gallery image fades in), image zoom effect (scale 1.1x on hover), “SELECT OPTIONS” button for variable products, “ADD TO CART” button for simple products, “READ MORE” button for out-of-stock products, price range display for variable products (e.g., “USD$605.00–USD$735.00”), sale badges with strikethrough pricing, and “OUT OF STOCK” badges.

The `enrichWithVariationInfo` helper was created to efficiently fetch variation counts and price ranges for product listings using a single SQL aggregation query with `MIN(CASE WHEN price > 0 THEN price END)` and `MAX(...)`.

Phase 8 — Category Page Fixes

Several category pages were showing “Category Not Found” or zero products due to slug mismatches between the frontend navigation and the database. The root cause was that WooCommerce uses long compound slugs (e.g., `samurai-swords-katanas-japanese-swords`) while the navigation used short slugs (e.g., `samurai-swords`).

The fix was a resilient four-tier slug matching fallback in `getCategoryBySlug`: (1) exact match, (2) starts-with match, (3) contains match, (4) name-based case-insensitive match. This resolved all category routing issues for Medieval Daggers, Samurai Swords, Axes, and Sale pages.

Phase 9 — Hover Effect Regression Fix

The product card hover effects (image swap, zoom, gold title) stopped working after a Tailwind CSS 4 update. Investigation revealed that Tailwind 4 wraps `group-hover:` utilities inside `@media (hover: hover)`, which does not trigger in all browser environments (including the Manus preview browser).

The fix was to replace all CSS-based hover effects with React state-driven effects using `onMouseEnter` / `onMouseLeave` handlers and inline styles. This approach bypasses the Tailwind limitation entirely and works universally across all browsers and environments.

Phase 10 — Additional Pages

Over 20 pages were built: Shop (with category sidebar, sorting, pagination), CategoryPage (with breadcrumbs), ProductPage (with variation selectors, attributes, gallery, specs, related products), CartPage, CheckoutPage (with full shipping/billing form), OrderConfirmation, AccountPage, WishlistPage, SearchPage, BlogPage, BlogPostPage, AboutPage, ContactPage, FAQPage, VideosPage, ReviewsPage, ShippingPage, PrivacyPage, TermsPage, and AdminDashboard (with order management, product management, and sales stats).

Session 2: PostgreSQL Conversion and Deliverable Package (Feb 14, 2026)

Phase 11 — PostgreSQL/Neon Conversion

The client requested deployment on Vercel with Neon PostgreSQL. The database was converted from MySQL to PostgreSQL with the following changes:

MySQL Construct	PostgreSQL Equivalent	Files Changed
`mysqlTable`	`pgTable`	`drizzle/schema.ts`
`mysqlEnum("role", [...])`	`pgEnum("user_role", [...])` then reference	`drizzle/schema.ts`
`int("id").autoincrement()`	`serial("id")`	`drizzle/schema.ts`
`int("field")`	`integer("field")`	`drizzle/schema.ts`
`json("field")`	`jsonb("field")`	`drizzle/schema.ts`
`timestamp().onUpdateNow()`	`timestamp()` (no auto-update in PG)	`drizzle/schema.ts`
`drizzle-orm/mysql2`	`drizzle-orm/node-postgres`	`server/db.ts`
`onDuplicateKeyUpdate`	`onConflictDoUpdate`	`server/db.ts`
`(result as any).insertId`	`.returning({ id: table.id })`	`server/db.ts`
`LIKE` (case-sensitive)	`ILIKE` (case-insensitive)	`server/db.ts`
`mysql2/promise`	`pg` (node-postgres)	`seed-db.mjs`
`connection.execute(sql, params)`	`client.query(sql, $1-style params)`	`seed-db.mjs`
`dialect: "mysql"`	`dialect: "postgresql"`	`drizzle.config.ts`

Column names were also converted from camelCase to snake_case to follow PostgreSQL conventions (e.g., `categoryId` → `category_id`, `stockStatus` →

`stock_status`). Drizzle ORM handles the mapping between TypeScript camelCase properties and database snake_case columns.

The schema was verified to generate correctly (12 tables, 0 errors) and migrations were tested against a local PostgreSQL instance. The `pg` and `@types/pg` packages were installed and `mysql2` was removed.

Phase 12 — Deliverable Package

The final deliverable was assembled as a zip file containing the complete source code (with PostgreSQL conversion applied), all 31 WooCommerce XML data files, extracted JSON data files, comprehensive documentation (README, Agent Replication Guide, Conversation Summary), and environment configuration templates.

Data Quality Fixes Applied

Throughout development, several data quality issues were discovered and fixed.

The first issue involved 36 products that had description text incorrectly stored in the weight field. This was caused by the XML extraction script matching overly broad patterns. The fix was to validate weight values and clear any that contained HTML or exceeded 50 characters.

The second issue involved 18 products with HTML entities (` `) in the weight field. These were cleaned by decoding HTML entities before storing.

The third issue involved short descriptions rendering raw HTML instead of formatted content. The fix was to use `dangerouslySetInnerHTML` in the React component to properly render the HTML content from WooCommerce.

The fourth issue involved 8 zero-price variable products. As described above, this was resolved by extracting prices from the variation data rather than the parent product record.

The fifth issue involved the category sidebar showing (0) counts for all categories. The fix was to compute product counts by iterating over the products table's `categories` jsonb array and counting occurrences of each category name.

Architecture Decisions Summary

Decision	Choice	Rationale
API framework	tRPC 11	End-to-end type safety, no manual REST routes needed
ORM	Drizzle	Lightweight, type-safe, generates SQL migrations
Styling	Tailwind CSS 4	Utility-first, matches template's design system
Routing	Wouter	Lightweight alternative to React Router
State management	React Context (CartContext)	Simple, sufficient for cart state
Image hover	React state + inline styles	Bypasses Tailwind 4 `@media(hover:hover)` limitation
Category matching	4-tier slug fallback	Handles WooCommerce's long compound slugs
Price display	SQL aggregation with CASE WHEN	Efficient single-query price range computation
JSON storage	PostgreSQL jsonb	Fast category filtering without joins
Auth	Manus OAuth	Built-in, zero-config authentication

What Remains To Be Done

The following items are not yet implemented and are documented for the next agent.

Payment Integration: The checkout form is built but Elavon Lightbox payment processing is not connected. This requires Elavon merchant credentials (Merchant ID, User ID, SSL PIN) from the client.

QuickBooks Integration: Automatic invoice creation is planned but requires QuickBooks API credentials (Client ID, Client Secret, Realm ID).

Neon Database Testing: The PostgreSQL conversion has been applied and verified at the schema level, but end-to-end testing with a real Neon database (seeding, queries, tests, UI) should be performed by the next agent. See the Agent Replication Guide for detailed testing instructions.

Image CDN Migration: Product images currently reference URLs on darksword-armory.com. For production reliability, these should be migrated to an independent CDN.

SEO: Meta tags, Open Graph tags, and JSON-LD structured data for product pages have not been implemented.