Darksword Armory — Full-Stack E-Commerce Clone

A pixel-perfect clone of darksword-armory.com, built with React 19, Express, tRPC, Drizzle ORM, and Tailwind CSS 4. This project replicates the original WooCommerce/Flatsome-powered site as a modern full-stack JavaScript application.

Table of Contents

Tech Stack

Layer	Technology
Frontend	React 19, Tailwind CSS 4, shadcn/ui, Wouter (routing)
Backend	Express 4, tRPC 11, Superjson
Database	Drizzle ORM (PostgreSQL dialect — Neon/Supabase/any PostgreSQL)
Auth	Manus OAuth (session cookies + JWT)
Build	Vite 6, esbuild, TypeScript 5.9
Testing	Vitest

> ***Database Dialect:*** *The codebase uses* ***PostgreSQL*** *via* `drizzle-orm/pg-core` *and the* `pg` *(node-postgres) driver. It is ready for deployment on* ***Neon***, ***Supabase***, *or any standard PostgreSQL 14+ database. For Neon serverless optimization, you can optionally swap the driver to* `@neondatabase/serverless` *— see the* [*Database Setup*]() *section.*

Project Structure

```
darksword_clone/
├── client/                     # Frontend (React + Vite)
│   ├── public/                 # Static assets
│   ├── src/
│   │   ├── _core/hooks/        # Auth hooks
│   │   ├── components/         # Reusable UI components
│   │   │   ├── ProductCard.tsx       # Product card with hover reveal
│   │   │   ├── ProductCarousel.tsx   # Featured products carousel
│   │   │   ├── CategoryShowcase.tsx  # Category grid with real images
│   │   │   ├── Header.tsx            # Navigation header
│   │   │   ├── Footer.tsx            # Site footer
│   │   │   ├── HeroSection.tsx       # Video hero with carousel
│   │   │   ├── CartDrawer.tsx        # Slide-out cart
│   │   │   └── ui/                   # shadcn/ui primitives
│   │   ├── contexts/           # React contexts (Cart, Theme)
│   │   ├── pages/              # Route-level page components
│   │   │   ├── Home.tsx              # Homepage
│   │   │   ├── Shop.tsx              # Shop with filters & sorting
│   │   │   ├── CategoryPage.tsx      # Category product listing
│   │   │   ├── ProductPage.tsx       # Product detail with variations
│   │   │   ├── CartPage.tsx          # Full cart page
│   │   │   ├── CheckoutPage.tsx      # Checkout form
│   │   │   ├── AdminDashboard.tsx    # Admin panel
│   │   │   └── ...                   # 20+ additional pages
│   │   ├── lib/trpc.ts         # tRPC client binding
│   │   ├── App.tsx             # Routes & layout
│   │   ├── main.tsx            # Providers & tRPC setup
│   │   └── index.css           # Global styles & theme
│   └── index.html
├── server/                     # Backend (Express + tRPC)
│   ├── _core/                  # Framework plumbing (OAuth, context, Vite)
│   ├── db.ts                   # Database query helpers
│   ├── routers.ts              # tRPC procedures (all API endpoints)
│   ├── storage.ts              # S3 file storage helpers
│   └── *.test.ts               # Vitest test files
├── drizzle/                    # Database schema & migrations
│   ├── schema.ts               # Table definitions (Drizzle ORM)
│   ├── relations.ts            # Table relations
│   └── *.sql                   # Migration files
├── shared/                     # Shared types & constants
├── data/                       # Source data for seeding
│   ├── products.json           # 367 products extracted from WooCommerce
│   ├── variations.json         # 1,150 product variations
│   ├── media.json              # Media library references
│   ├── posts.json              # Blog posts
│   ├── pages.json              # Static pages
│   └── xml/                    # Original WordPress XML exports (31 files)
├── seed-db.mjs                 # Database seeding script
├── drizzle.config.ts           # Drizzle Kit configuration
├── vite.config.ts              # Vite build configuration
├── vitest.config.ts            # Test configuration
├── package.json                # Dependencies & scripts
└── todo.md                     # Feature tracking
```

Prerequisites

- **Node.js** 20+ (22.x recommended)
- **pnpm** 9+ (package manager)
- **PostgreSQL 14+** database (Neon, Supabase, or local PostgreSQL)

Local Development Setup

```
# 1. Clone the repository
git clone <your-repo-url> darksword_clone
cd darksword_clone

# 2. Install dependencies
pnpm install

# 3. Create .env file (see Environment Variables section)
cp .env.example .env
# Edit .env with your database URL and other credentials

# 4. Push database schema
pnpm db:push

# 5. Seed the database
node seed-db.mjs

# 6. Start development server
pnpm dev
```

The dev server runs on `http://localhost:3000` by default.

Database Setup

Option A: Neon PostgreSQL (Recommended for Production)

1. Create a free account at neon.tech
2. Create a new project and database
3. Copy the connection string and set it in `.env`:

```
DATABASE_URL=postgresql://username:password@ep-xxxxx.us-east-
2.aws.neon.tech/neondb?sslmode=require
```

1. Push the schema and seed:

```
pnpm db:push
node seed-db.mjs
```

Optional: Neon Serverless Driver

For optimal performance on Vercel serverless functions, you can swap the standard `pg` driver for `@neondatabase/serverless`:

```
pnpm add @neondatabase/serverless
```

Then update `server/db.ts`:

```
import { drizzle } from "drizzle-orm/neon-http";
import { neon } from "@neondatabase/serverless";

const sql = neon(process.env.DATABASE_URL!);
const db = drizzle(sql);
```

Option B: Local PostgreSQL

For local development:

```
sudo apt install postgresql
sudo -u postgres createdb darksword
```

Set in `.env`:

```
DATABASE_URL=postgresql://postgres:postgres@localhost:5432/darksword
```

Seeding the Database

The `seed-db.mjs` script reads from `data/products.json` and `data/variations.json` to populate:

- **62 categories** with slugs and hierarchy
- **367 products** with full metadata (descriptions, specs, images, gallery)
- **1,150 product variations** (Package options, sizes, models, colors)
- **503 product attributes** (Grip colors, Scabbard, Blade Finish, Guard & Pommel Finish)

```
# After schema is pushed
node seed-db.mjs
```

> ***Important:*** *The seed script references `data/products.json` and `data/variations.json`. Make sure these files are in the `data/` directory relative to the project root. Update the file paths in `seed-db.mjs` if needed.*

Environment Variables

Create a `.env` file in the project root:

```
# Required
DATABASE_URL=postgresql://username:password@ep-xxxxx.us-east-
2.aws.neon.tech/neondb?sslmode=require

# Authentication (Manus OAuth — required for login)
VITE_APP_ID=your_app_id
JWT_SECRET=your_jwt_secret
OAUTH_SERVER_URL=https://api.manus.im
VITE_OAUTH_PORTAL_URL=https://auth.manus.im

# Owner info
OWNER_OPEN_ID=your_open_id
OWNER_NAME=your_name

# Built-in APIs (optional — for LLM, image gen, notifications)
BUILT_IN_FORGE_API_URL=https://api.manus.im
BUILT_IN_FORGE_API_KEY=your_api_key
VITE_FRONTEND_FORGE_API_KEY=your_frontend_key
VITE_FRONTEND_FORGE_API_URL=https://api.manus.im

# App branding
VITE_APP_TITLE=Darksword Armory
VITE_APP_LOGO=/darksword-logo.svg
```

Deployment to Vercel

Step 1: Push to GitHub

```
git init
git add .
git commit -m "Initial commit: Darksword Armory clone"
git remote add origin https://github.com/YOUR_USERNAME/darksword-armory.git
git push -u origin main
```

Step 2: Import to Vercel

1. Go to vercel.com/new
2. Import your GitHub repository

3. Set the following build settings:
    - **Framework Preset:** Other
    - **Build Command:** `pnpm build`
    - **Output Directory:** `dist`
    - **Install Command:** `pnpm install`

Step 3: Configure Environment Variables

Add all environment variables from the `.env` section above in the Vercel dashboard under **Settings → Environment Variables**.

Step 4: Configure Vercel for Express Backend

Create a `vercel.json` in the project root:

```json
{
  "version": 2,
  "builds": [
    {
      "src": "dist/index.js",
      "use": "@vercel/node"
    }
  ],
  "routes": [
    {
      "src": "/(.*)",
      "dest": "dist/index.js"
    }
  ]
}
```

> ***Note:*** *Vercel's serverless functions have a 10-second timeout on the free tier. For long-running operations (image generation, LLM calls), consider upgrading to Pro or using a different hosting provider for the backend.*

Features Implemented

E-Commerce Core

- Full product catalog with 367 products across 62 categories
- Product variations (1,150 total): Package options, sizes, models, colors
- Product attributes: Grip color swatches, Scabbard colors, Blade Finish dropdowns, Guard & Pommel Finish dropdowns
- Shopping cart with variation support (add, remove, update quantity)
- Checkout form with shipping/billing addresses
- Order management system
- Wishlist functionality

Product Display (Matching Original Site)

- **Image hover swap**: Second gallery image fades in on hover (React state-driven)
- **Image zoom**: Scale 1.1x on hover
- **Product cards**: Info below image, category label, price ranges for variable products
- **Action buttons**: “SELECT OPTIONS” for variable products, “ADD TO CART” for simple products
- **Sale badges** and **OUT OF STOCK** badges
- **Price ranges** for variable products (e.g., “USD$605.00–USD$735.00”)

Product Page

- Image gallery with thumbnails
- Package variation selector (dropdown with prices)
- Grip color swatches (6 colors)
- Scabbard color swatch
- Blade Finish dropdown (High polish / Satin finish)
- Guard & Pommel Finish dropdown (Highly polished / Satin Finish)

- Product specifications table
- Related products section
- Tabbed content (Description / Additional Info / Reviews)

Pages (20+)

- Homepage with video hero, product carousel, category showcase
- Shop page with category sidebar, sorting, pagination
- Category pages with breadcrumbs
- Product detail pages
- Cart page and checkout
- Order confirmation
- Account page
- Wishlist page
- Search page
- Blog and blog post pages
- About, Contact, FAQ, Videos, Reviews pages
- Shipping, Privacy, Terms pages
- Admin dashboard (orders, products, sales stats)

Design

- Dark theme with gold (#C8A45C) accents matching original site
- Cinzel Decorative + Cormorant Garamond typography
- Responsive design (mobile-first)
- Announcement bar with rotating messages
- Sticky header with navigation
- Footer with newsletter signup

Architecture Overview

Data Flow

```
Client (React) → tRPC hooks → Express/tRPC server → Drizzle ORM → PostgreSQL
(Neon)
```

Key Design Decisions

1. **tRPC for API**: End-to-end type safety between frontend and backend. No REST routes or manual type definitions needed.
2. **Drizzle ORM**: Lightweight, type-safe ORM that generates SQL migrations. Schema-first approach with `drizzle-kit`.
3. **Product Variations vs Attributes**:
   - **Variations** (productVariations table) affect price — Package options like "Blunt blade & Scabbard: $580"
   - **Attributes** (productAttributes table) are cosmetic — Grip colors, Blade Finish, etc.
4. **Image Hover Effect**: Uses React `onMouseEnter`/`onMouseLeave` state instead of CSS `:hover` because Tailwind 4 wraps `group-hover:` in `@media (hover: hover)` which doesn't work in all environments.
5. **Category Slug Matching**: Resilient partial-match fallback for category slugs (e.g., "samurai-swords" matches "samurai-swords-katanas-japanese-swords").

Scripts

Command	Description
`pnpm dev`	Start development server with hot reload
`pnpm build`	Build for production (Vite + esbuild)
`pnpm start`	Run production server
`pnpm test`	Run Vitest test suite (24 tests)
`pnpm db:push`	Generate and run database migrations
`pnpm check`	TypeScript type checking
`pnpm format`	Format code with Prettier

License

This project is a clone built for educational/development purposes. All product data, images, and branding belong to Darksword Armory Inc.