Project Specification: Hardware Store

E-Commerce & Services Platform

1. Overview

A full-stack e-commerce web application for a hardware store selling products in categories like construction, welding, plumbing, electricals, tools, mabati (sheet metal), woodwork, paint, fittings, nails, and related items. It also supports on-request services (e.g., transport/delivery, installation/assembly, cutting). Users can browse catalogs, manage carts/wishlists, request services/quotes, and complete checkouts. Admins handle products, inventory, orders, services, users, and reporting.
The system prioritizes DRY principles by reusing components (e.g., shared UI elements, API handlers) and SOLID principles through modular design: single-responsibility modules (e.g., separate auth, catalog, order services), open-closed extensibility (e.g., pluggable payment providers), Liskov-substitutable interfaces (e.g., uniform variant handling), interface segregation (e.g., role-specific APIs), and dependency inversion (e.g., injectable repositories via interfaces in Go).
Tech Stack :
● **Frontend** : Next.js (App Router), TypeScript, Tailwind CSS, shadcn/ui, TanStack Query for data fetching, Zustand for state management, Server Actions for mutations.
● **Backend** : Go (Golang) for RESTful APIs, PostgreSQL for relational data, Redis for caching/sessions.
● **Infra/DevOps** : Docker for containerization, GitHub Actions for CI/CD, environment-based config, cloud object storage/CDN for images. Digital Ocean VPS.

2. Goals

Goals :
● Robust e-commerce flows: auth, catalog search/filter, cart/wishlist, checkout/payments, orders/returns/invoices.
● Service requests & scheduling (e.g., transport): capture details, auto/manual quoting, assignment.
● Role-based admin portal for catalog/inventory/pricing/orders/services/content/reporting.
● Scalable, secure, performant architecture with auditability and observability.

PROF

3. User Roles

```
● Visitor (Guest) : Basic browsing, search, and cart (session-based).
● Customer (Registered) : Personalized features like persistent
cart/wishlist, order history,
service requests.
```

```
● Admin (Super Admin) : Full management access with role guards.
```

4. Core Features & User Stories

4.1 Authentication

● Registration/login with email/password/phone.
● Password reset via email/SMS.
● JWT-based sessions with refresh/rotation.
● Logout and session invalidation.
User Stories :
● As a visitor, I can register to access personalized features.
● As a user, I can log in to view my profile/orders.
● As a user, I can reset my password if forgotten.
● As a user, I can log out securely.

4.2 Catalog & Search

● Hierarchical categories (e.g., construction > tools).
● Product details with variants (size/color/gauge), units, pricing tiers, bulk discounts, images/specs/related items.
● Search with autocomplete; filters (brand/price/availability/specs), sorting, pagination.
User Stories :
● As a visitor/user, I can view homepage with featured products/categories.
● As a visitor/user, I can browse by category.
● As a visitor/user, I can search by keywords.
● As a visitor/user, I can filter/sort products.
● As a visitor/user, I can view product details.

4.3 Cart, Wishlist, Checkout

PROF

● Persistent cart (guest via session, merged on login).
● Wishlist for saving items.
● Checkout: Addresses, delivery/pickup options, service selection, fee estimates, taxes/coupons, payment (card/M-Pesa/bank/Payment on delivery).
● Order confirmation with PDF invoice, email/SMS notifications.
User Stories :
● As a user, I can add/edit/remove cart items.
● As a user, I can manage wishlist and move items to cart.
● As a user, I can proceed to checkout, enter details, select services.
● As a user, I can complete payment and track orders.

4.4 Services

● Request form: Type (transport/installation), locations, dates, item details, instructions.
● Quote flow: Auto-estimate (distance/weight) + manual override.

● Scheduling: Assign driver/vehicle, status updates (Requested > Quoted > Accepted > Scheduled > In-Progress > Completed > Billed).

User Stories :

● As a user, I can request services during/after checkout.

● As a user, I can view/accept quotes and track status.

4.5 Customer Account

● Profile editing, addresses, saved payments (tokenized), order history, invoices, service requests, returns, wishlist.

User Stories :

● As a user, I can view/update profile and history.

4.6 Admin Portal

● CRUD for products/categories/attributes/variants/media/SEO.

● Inventory: Stock levels/adjustments, suppliers/purchase orders, low-stock alerts.

● Pricing: Base/sale/tiered, coupons.

● Orders: Status updates, refunds/returns, shipments.

● Services: Quotes/assignments/calendar, completions.

● Users/roles management; content (banners/pages); settings.

● Reports: Sales/top products/inventory/payments/

● Audit logs for actions.

User Stories :

● As an admin, I can access dedicated panel.

● As an admin, I can manage products/inventory/orders/users/services.

● As an admin, I can view analytics/reports.

4.7 General

● Responsive design (mobile/desktop).

● Error handling (e.g., out-of-stock messages).

● Basic SEO (structured data, sitemaps).

PROF

User Stories :

● As a user/admin, the site is responsive.

● As a user, I see clear error messages.

5. Non-Functional Requirements

```
● Security : HTTPS/HSTS, CSRF protection, JWT/Argon2id hashing, OWASP
mitigations,
input validation, GDPR-like data rights.
● Performance : CDN caching, DB indexing, ETag, pagination, async queues
for tasks
(emails/PDFs/notifications).
● Scalability : Modular services, Redis for sessions/cache, horizontal
scaling.
```

```
● Accessibility/i18n/SEO : WCAG 2.1 AA, keyboard nav, alt text; en-KE
locale, KES currency,
VAT; structured data/canonicals.
● Observability : Metrics/logs/traces, health checks, rate-limiting,
backups.
```

6. Architecture

```
● Frontend : ISR/SSG for static catalog, SSR for dynamic (cart/account),
client-side
interactivity.
● Backend : REST APIs with validation/OpenAPI specs; repositories via
interfaces for DI.
● Data : PostgreSQL (core), Redis (transient), object storage (media).
● Flows : Catalog via API fetches; cart mutations via Server
Actions/APIs; orders reserve
inventory atomically; services use queues for notifications.
```

7. Data Model (PostgreSQL Schema)

The schema is simplified for clarity and minimalism, avoiding over-normalization while ensuring DRY (no redundant fields) and SOLID (single-responsibility tables, extensible via JSONB for flexibility). It supports core e-commerce (products, carts, orders) and services (e.g., transport).

● users:

○ Columns: id (uuid, pk), email (varchar, unique), password_hash (varchar), phone (varchar, nullable), full_name (varchar), role (enum: 'customer', 'admin'), created_at (timestamp).

○ Notes: Single table for users; role enum simplifies access control; no updated_at to reduce complexity.

● addresses:

PROF

○ Columns: id (uuid, pk), user_id (uuid, fk to users), label (varchar, e.g., 'Home'), line (varchar), city (varchar), country (varchar), is_default (boolean, default false).

○ Notes: Minimal address fields; separate table for reusability across orders/services.

● categories:

○ Columns: id (uuid, pk), name (varchar, e.g., 'Construction'), slug (varchar, unique).

○ Notes: Flat categories (no hierarchy) for simplicity; slug for SEO-friendly URLs.

● products:

○ Columns: id (uuid, pk), sku (varchar, unique), name (varchar), slug (varchar, unique), category_id (uuid, fk to categories), description (text), price (decimal), stock_quantity (int), images_json (jsonb, e.g., ['url1', 'url2']).

○ Notes: Combines product and variant data

```
● carts:
○ Columns: id (uuid, pk), user_id (uuid, fk to users, nullable),
session_id (varchar,
```

```
unique, nullable).
○ Notes: Supports guest carts via session_id; no updated_at to simplify.
● cart_items:
○ Columns: id (uuid, pk), cart_id (uuid, fk to carts), product_id (uuid,
fk to products),
quantity (int), unit_price (decimal).
○ Notes: Minimal fields; unit_price snapshots price at add time; total
computed
dynamically.
● wishlists:
○ Columns: id (uuid, pk), user_id (uuid, fk to users), product_id (uuid,
fk to products).
○ Notes: Single table for wishlist items (no separate wishlist table) to
reduce
complexity.
● orders:
○ Columns: id (uuid, pk), user_id (uuid, fk to users, nullable), total
(decimal), status
(enum: 'pending', 'confirmed', 'shipped', 'delivered'), address_json
(jsonb),
service_request (jsonb, nullable, e.g., {'type': 'transport', 'details':
'10km delivery'}),
placed_at (timestamp).
○ Notes: address_json snapshots address; service_request embeds service
details to
avoid separate table; status enum for simplicity.
● order_items:
○ Columns: id (uuid, pk), order_id (uuid, fk to orders), product_id
(uuid, fk to
products), quantity (int), unit_price (decimal).
○ Notes: Snapshots product price; total computed dynamically.
● payments:
○ Columns: id (uuid, pk), order_id (uuid, fk to orders), provider
(varchar, e.g.,
'M-Pesa'), reference (varchar, unique), amount (decimal), status (enum:
'pending',
'completed'), paid_at (timestamp, nullable).
○ Notes: Minimal payment tracking; no raw_response_json to simplify.
● notifications:
○ Columns: id (uuid, pk), user_id (uuid, fk to users, nullable), channel
(enum: 'email',
'sms'), message (text), status (enum: 'pending', 'sent'), sent_at
(timestamp, nullable).
○ Notes: Simplified notifications; message replaces
template/payload_json.
```

PROF

8. API Endpoints (REST Examples)

Endpoints are simplified to cover essential actions, using consistent patterns (DRY) and role-based guards (SOLID). Only core resources and actions are included to minimize complexity.

● **Auth** :

○ POST /api/auth/register: Create user.
○ POST /api/auth/login: Return JWT.
○ POST /api/auth/logout: Invalidate session.
○ POST /api/auth/password/reset: Request/reset password.
● **Catalog** :
○ GET /api/categories: List categories.
○ GET /api/products?category=slug&q=term: Search products (basic filters).

```
○ GET /api/products/:slug: Get product details.
● Cart/Wishlist :
○ GET /api/cart: Get cart.
○ POST /api/cart/items: Add item.
○ PUT /api/cart/items/:id: Update quantity.
○ DELETE /api/cart/items/:id: Remove item.
○ GET /api/wishlist: Get wishlist.
○ POST /api/wishlist/items: Add item.
○ DELETE /api/wishlist/items/:id: Remove item.
● Checkout/Orders :
○ POST /api/checkout/place: Create order with address/service.
○ GET /api/orders: List user orders.
○ GET /api/orders/:id: Get order details.
● Payments :
○ POST /api/payments/initiate: Start payment.
○ POST /api/payments/webhook: PSP callback.
● Admin (Guarded) :
○ GET/POST/PUT/DELETE /api/admin/categories: Manage categories.
○ GET/POST/PUT/DELETE /api/admin/products: Manage products.
○ PUT /api/admin/inventory: Update stock.
○ GET/PUT /api/admin/orders/:id: Manage orders.
○ GET /api/admin/reports: Basic sales/stock report.
```

9. Frontend Routes (Next.js)

PROF

Routes are minimal, using Next.js App Router for ISR/SSR. DRY is achieved via shared layouts; SOLID via focused page responsibilities.
● /: Home (ISR, featured products/categories).
● /search: Search with basic filters (SSR).
● /c/[slug]: Category listing (ISR).
● /p/[slug]: Product detail (ISR).
● /cart: Cart view (SSR, Server Actions).
● /wishlist: Wishlist (SSR).
● /checkout: Checkout (address/service/payment, SSR).
● /account: Profile/orders (SSR).
● /admin: Protected dashboard (SSR, sub-routes: catalog/inventory/orders/reports).
● /pages/[slug]: CMS pages (SSG, e.g., About).

10. Integrations

- Payments: lets use paystack
- Messaging: Twilio for SMS/WhatsApp; SendGrid for email.
- Storage: cloudinary.
- Analytics: Privacy-friendly web tracking; built-in sales dashboards.