eCommerce AI

A local commerce prototype that brings catalog, stock, orders and pricing experiments into one interface for store operators and project reviewers.

THE PROBLEM

Store information is hard to evaluate when products, stock and orders are disconnected. In this prototype, empty lists also made the dashboard and pricing tools impossible to demonstrate meaningfully.

THE SOLUTION

Use one React interface and one local API, backed by linked SQLite records. Synthetic demo data now fills the operational screens and supports a repeatable, transparent pricing walkthrough.

How it works

React admin UI
Catalog, dashboard, orders

→

Express / Node API
CRUD and pricing rules

→

SQLite database
Products, users, order items

The FastAPI/Django alternatives and shared Python AI modules are separate | examples, outside this running demo.

Scope of the current demonstration

Catalog & inventory	View products, prices, categories and stock; create, edit and delete products.
Orders & dashboard	Inspect orders and line items; compute product/order counts and total order value.
Pricing sandbox	Preview a price using demand, stock and competitor inputs; the catalog price is unchanged.
Demo accounts & data	Registration/basic login APIs and persistent sample records; prototype sign-in UI.

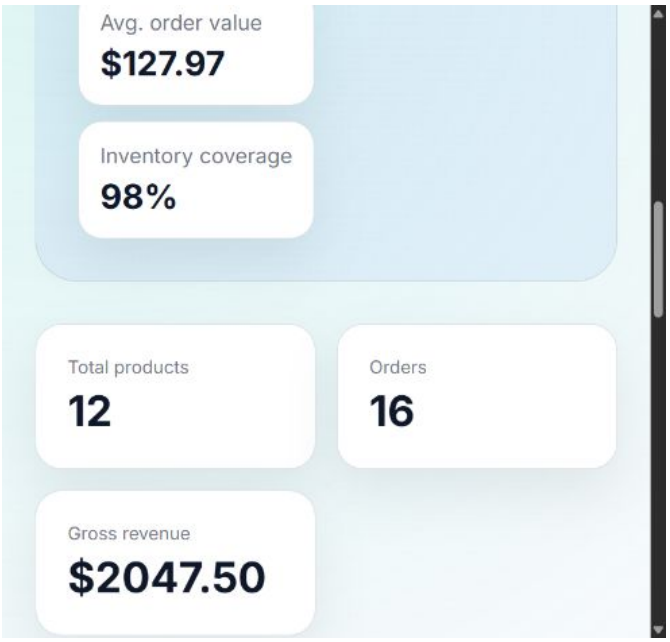
A store you can explore

All people, purchases and reviews are fictional. The dataset includes six product categories, 36 linked order items, and 10 stored reviews (reviews have no screen in the current UI).

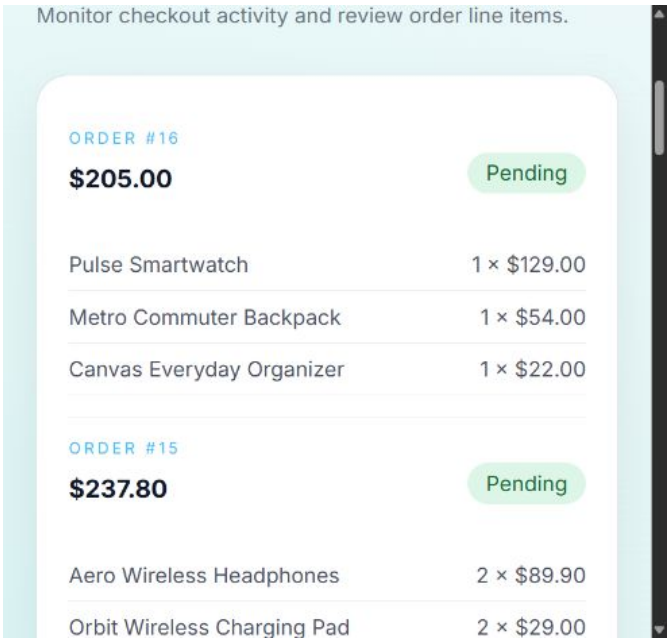
12	16	9	$2,047.50
PRODUCTS	ORDERS	DEMO ACCOUNTS	ORDER VALUE

Average order value: **$127.97**. The total includes every order status; it is sample order value, not settled payments.

Real screens from the populated demo



Dashboard: counts and order value come from SQLite through the API. The displayed 98% coverage is a placeholder.



Order #16: smartwatch $129 + backpack $54 + organizer $22 = $205. The screen shows quantities and fulfillment status.

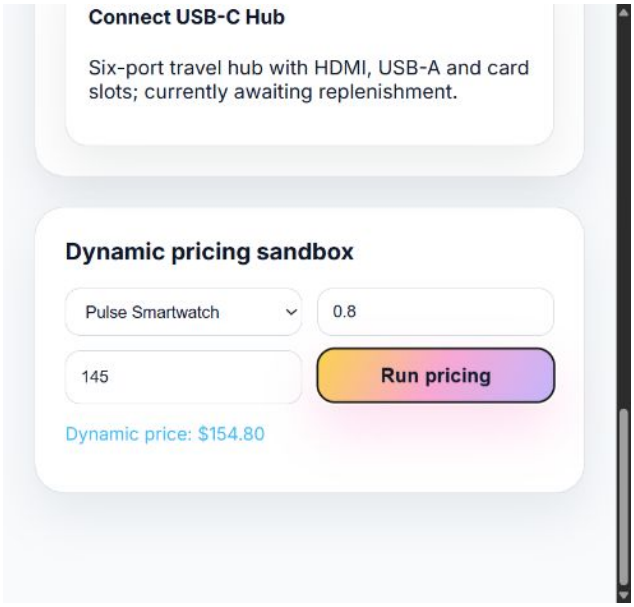
Scenarios worth demonstrating

Stock pressure: Pulse Smartwatch has 3 units; Connect USB-C Hub has 0. **Fulfillment:** 8 delivered, 4 shipped, 2 processing and 2 pending orders. **Accounts:** one demo admin and eight synthetic customers.

```
RESEED  python tools/seed_demo_data.py  (safe to rerun)
```

Pricing, fixes & next steps

The pricing tool exposes a simple rule so reviewers can trace the result. It previews a suggestion; it does not update the product or place an order.



PULSE SMARTWATCH

Base $129 | Demand 0.8
Stock 3 | Competitor $145

Demand adjustment	$149.64
Low-stock uplift (+10%)	$164.604
Blend with competitor	$154.802
Rounded preview	**$154.80**

Observed result from the live pricing sandbox. The base rule is price x (1 + 0.2 x demand), followed by stock and competitor adjustments.

What was fixed for this demo

Startup: rebuilt the missing SQLite native binding and started UI/API. **Empty screens:** seeded linked data with a pre-seed backup and duplicate protection. **Readability:** corrected catalog title, price and label contrast in light mode.

Current boundaries and the work ahead

Access	Sign-in UI bypasses API validation; stored passwords are plaintext and routes lack authorization. Next: hashed passwords, sessions and enforced roles.
Intelligence	Recommendations return the newest products, including zero-stock items. Pricing is rule-based. Next: stock-aware recommendations and integrated AI modules.
Commerce & metrics	Cart/checkout, payments and shipping integrations are absent. Conversion, coverage and pulse are placeholders. Next: complete checkout and measured analytics.

Verified: SQLite integrity, foreign keys, matching order totals, zero inserts on a second seed run, frontend build (legacy ESLint plugin disabled), live screens and $154.80 pricing. Evidence: Node controllers/routes and React components; details in docs/demo-guide.md.