

Technical Architecture Documentation — *Focal Shop*

1. Project Overview

Focal Shop is a full-stack e-commerce web application built to enable users to:

- Browse a catalog of products.
- View detailed information for each product.
- Add/remove items to/from a shopping cart.
- View the total cart value dynamically.

The architecture separates the frontend (React) and backend (Node.js/Express) layers, with product data stored in MongoDB Atlas (or mock JSON for development). The UI is responsive and minimal, using Tailwind CSS.

| Layer | Technology | Purpose |
|---|--------------------------|--|
| Frontend | React.js | SPA UI framework |
| | React Router DOM | Client-side routing |
| | Tailwind CSS | Utility-first styling & responsiveness |
| | Axios | HTTP requests to backend |
| | Context API (or similar) | Manage cart state globally |
| Backend | Node.js + Express.js | REST API server |
| | Mongoose (optional) | Object modelling for MongoDB |
| | dotenv | Environment configuration |
| | CORS | Cross-origin resource sharing |
| Database | MongoDB Atlas | Product data persistence |
| Deployment (Assumed) Vercel / Lovable.app / Render Hosting frontend + backend | | |

4. Modules / Components

Frontend Modules

- **Navbar**: global navigation, shows cart item count.
- **Home / Products List Page**: fetches /api/products, displays product cards.

- **Product Details Page:** fetches /api/products/:id, shows name, price, description, image, “Add to Cart”.
- **Cart Page:** shows items in cart, allows removal or quantity update, shows dynamic total.
- **Cart Context:** holds cart state (items: [{ productId, name, price, image, quantity }]), exposes methods (addItem, removeItem, updateQty, clearCart).
- **API Service Layer:** centralised functions for fetchProducts(), fetchProductById(id), maybe calculateCartTotal(items).

Backend Modules

- **Products Route** (/api/products):
 - GET /api/products → list all products
 - GET /api/products/:id → product details
- **Cart Utility Route** (/api/cart):
 - POST /api/cart/calculate → receives list of items, returns computed total (useful for client/server sync)
- **Data Source:** Either real MongoDB collection (products) or fallback products.json for mock.
- **Models:** Product schema with fields name, price, description, image.

5. Data Flow

1. User visits frontend → Home page loads.
2. Frontend calls GET /api/products.
3. Backend fetches product list (from DB or JSON), returns JSON.
4. Frontend renders list of product cards.
5. User clicks a product → Frontend navigates to /products/:id.
6. Frontend calls GET /api/products/:id, backend responds with product details.
7. User clicks “Add to Cart” → Frontend cart context updates state (adds item, quantity).
8. User visits “Cart” page → Cart items show, quantity adjustable, remove option.
9. Optionally frontend sends POST /api/cart/calculate with items to validate total on server side.
10. Total displayed to user (local + optional server result).
11. For production extension, checkout flow would go further: creating orders, payment gateway, etc.

6. Database Schema

Collection: `products`

| Field | Type | Description |
|--------------------------|-----------------------|------------------------------|
| <code>_id</code> | <code>ObjectId</code> | Unique ID |
| <code>name</code> | <code>String</code> | Product title |
| <code>price</code> | <code>Number</code> | Product price |
| <code>description</code> | <code>String</code> | Detailed product description |
| <code>image</code> | <code>String</code> | URL to product image |