Prompt Engineering

Technical Architecture Documentation

- You are an expert system architect. I am building an e-commerce site with React (frontend), Node.js + Express (backend), and MongoDB. The site must let users browse products, view product details, add/remove items to a cart, update quantities, and checkout flow placeholder (no payment integration required now). Users can be anonymous (guest cart) or authenticated. Provide a clear list of functional and non-functional requirements, constraints (budget, hosting, timeline), and success metrics. Also list open questions I should answer.
- Using the requirements above, produce a high-level architecture document: components, data flow, responsibilities, and a block diagram. Include services/components: React SPA, Node/Express API, MongoDB (atlas or self-hosted), auth service (JWT), cache layer (optional), CDN, and deployment architecture (Vercel/Netlify + Node host or single container). Explain trade-offs for using MongoDB, JWT vs session cookies, and client-side state options (Context API vs Redux). Return a mermaid diagram I can paste into docs.

Code Generation

- Produce MongoDB document schemas (Mongoose models) for: User, Product, Category, Cart, Order (placeholder). For each model include fields, types, indexes, example documents, and reasoning for indexes. Also provide sample seed data (10 products) in JSON.
- Generate an OpenAPI 3.0 (YAML) spec for the REST API with endpoints:
  - GET /api/products (list + pagination + filters)
  - GET /api/products/{id} (details)
  - POST /api/cart (create/guest cart)
  - GET /api/cart/{id}
  - PUT /api/cart/{id}/items (add/update items)
  - DELETE /api/cart/{id}/items/{itemId}
  - POST /api/users/register, POST /api/users/login
  - GET /api/users/me
  - POST /api/orders (placeholder)

  Include request/response schemas, example payloads, auth requirements (JWT), and error responses.
- Add a security section: authentication (JWT best practices), storing passwords (bcrypt), secure cookie vs localStorage tradeoffs for tokens, input validation, rate limiting, CORS, helmet, CSP, OWASP checklist items, and data privacy (what to store and what not). Include sample Express middleware code for helmet, rate-limit, and input sanitization.

Generate a project scaffold for a monorepo named "shop-app" with these folders:
- /packages/backend (Node.js + Express + Mongoose)

- /packages/frontend (React, Vite or Create React App)
- /packages/shared (Types or DTOs)
- /scripts (seed, dev helper)

Provide package.json files and recommended npm scripts for dev, lint, test, build, start, and a README that explains how to run locally with env vars. Prefer JavaScript (or TypeScript if I specify).

Backend — Express API

Create a production-ready Express backend in /packages/backend that includes:
- Mongoose models (from earlier)
- Auth (register/login) using bcrypt + JWT, with refresh tokens optional
- Controllers for products, cart, users, orders (orders can be placeholder)
- Validation using Joi or express-validator
- Error handling middleware
- Logging (morgan + winston)
- Environment config pattern (.env.example)
- Scripts: start, dev (nodemon), lint, test

Return all files: server.js/app.js, routes, controllers, models, middleware, package.json.

Frontend — React SPA

Create a React frontend (Vite recommended) in /packages/frontend with:
- Folder structure: src/components, src/pages, src/hooks, src/services, src/state
- Global state for cart using React Context + useReducer (include persistence to localStorage for guest cart)
- Pages: Home (product grid), ProductDetails, Cart, Checkout (placeholder), Login/Register, Profile
- ProductCard, ProductList, Header with cart icon and item count, Responsive layout, basic accessible components
- API service that consumes the backend OpenAPI endpoints
- Unit tests for major components (React Testing Library + Jest)
- Tailwind or CSS module styling with simple, modern UI (no external paid libs)

Return all files including vite config, package.json, and instructions to run.

Implement cart handling that supports:
- Guest cart persisted in localStorage with a unique cartId

- When user logs in, merge guest cart with server-side cart (resolve duplicates by sum of quantities)

Provide backend endpoints and frontend logic for this workflow, with example sequence of API calls and example merged cart logic.