E-commerce Assignment

Architecture Documentation

Team

November 10, 2025

Contents

1 System Overview

This system is a minimal full-stack e-commerce demo with:

- **Frontend**: React (Vite), client-side search/filter, cart state, JWT persistence, toast notifications.
- **Backend**: Node.js + Express, JWT authentication, in-memory data for users and products, protected checkout.
- **Data**: Stored in-memory (no external DB). Products enriched with realistic categories and images.

Primary goals are a clean demo-ready UI/UX, add-to-cart toasts, quantity adds from product cards, and simple checkout with JWT.

2 High-Level Architecture

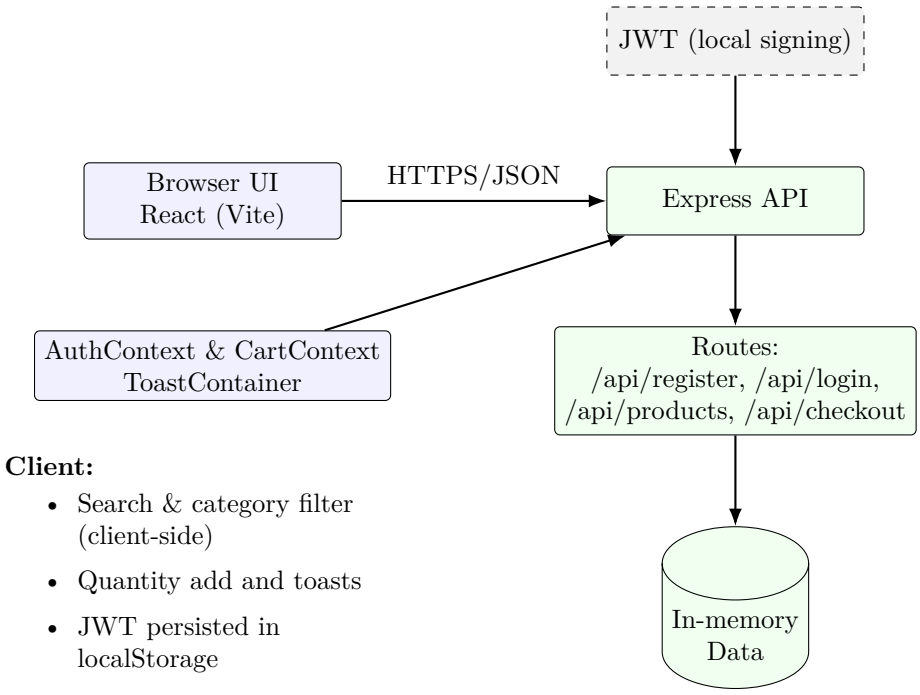

Figure 1: High-level architecture and data flow.

3 Components

3.1 Frontend (React/Vite)

- `App.jsx`: App shell, routes, header, global `ToastContainer`.
- `AuthContext.jsx`: Login/register, token persistence in localStorage.

- `CartContext.jsx`: Cart items, aggregation (increment quantities), totals/count.
- `Products.jsx`: Fetches products, client-side search (debounced) and category filter, calls `addToCart`.
- `ProductCard.jsx`: Shows image, title, description, category, price, quantity input (Enter to add).
- `ToastContainer.jsx`/`Toast.jsx`: Generic toasts API (`addToast({message,durationMs})`).

3.2 Backend (Node/Express)

- `src/routes/auth.js`: `/api/register`, `/api/login`; issues JWTs.
- `src/routes/products.js`: `/api/products`, `/api/products/:id`.
- `src/routes/checkout.js`: `/api/checkout` (JWT required), computes totals server-side.
- `src/middleware/auth.js`: JWT verification, `authRequired`.
- `src/data/products.js`: In-memory product catalog with categories & images.
- `src/data/users.js`: In-memory users, demo user seeding, bcrypt hashing.

4 API Surface

Endpoint	Description	Auth
`POST /api/register`	Body: `{name, email, password}` → `{user, token}`	No
`POST /api/login`	Body: `{email, password}` → `{user, token}`	No
`GET /api/products`	Returns full product list with categories and images	No
`GET /api/products/:id`	Returns product details by id	No
`POST /api/checkout`	Body: `{ items: [{productId, quantity}] }` → `{ success, order }`	Yes

5 Authentication and Authorization

- **JWT**: Signed server-side using a shared secret. Expiration ≈ 7 days.
- **Client**: Token stored in localStorage via `AuthContext`.
- **Protected routes**: `/api/checkout` uses `Authorization: Bearer <token>` header.

6 Frontend Behavior

6.1 Search and Filtering

- Debounced (300 ms) text search across title and description.
- Category dropdown across: Phones, Laptops, Accessories, Keyboards, Monitors, Tablets, Wearables, Home.
- Filters are client-side only; backend is unchanged.

6.2 Cart & Toasts

- `CartContext.addToCart(product, qty)` aggregates quantities; no duplicate lines.
- Product cards allow quantity entry; Enter key or Add button triggers add.
- `ToastContainer` shows top-right auto-dismissing toasts (2.5s).

7 Data Model (In-Memory)

- **Product**: `{ id, title, description, price, category, image }`
- **User**: `{ id, name, email, passwordHash, createdAt }`
- **Order (response only)**: `{ id, userId, items:[{productId,title,price,quantity,lineTotal}], total, createdAt }`

8 Sequence (Checkout)

1. User fills cart and clicks Checkout (frontend).
2. Frontend sends `POST /api/checkout` with JWT.
3. Backend validates JWT, verifies items & prices from in-memory products.
4. Backend returns synthesized `order` with line items and total.
5. Frontend clears cart and shows success toast/message.

9 Non-Functional Concerns

9.1 Security

- Passwords hashed with bcrypt before storing in-memory.
- JWT secret provided via environment variable; do not commit secrets.

9.2 Performance

- Lightweight; in-memory data access is O(1)/O(n) as needed.
- Client-side filtering avoids additional network calls.

10 Local Development

- Backend: `npm run start` (default `http://localhost:5000`)
- Frontend: `npm run dev` (default `http://localhost:5173`)
- Set `VITE_API_URL` to point frontend at backend if needed.

11 Future Improvements

- Replace in-memory stores with a database (e.g., Postgres/Mongo).
- Images CDN and responsive sizes.
- Payment integration and orders persistence.
- E2E tests for auth, cart, and checkout flows.