Technical Architecture

🏗️ High-Level Architecture Document

1. Overview

The e-commerce application follows a **three-tier architecture**:

1.
2. **Frontend (React SPA)** – user interface, routing, and client-side state management.
3.
4. **Backend (Node.js + Express API)** – RESTful services handling business logic, authentication, and database communication.
5.
6. **Database (MongoDB)** – stores all persistent data such as products, users, carts, and orders.
7.

Additional supporting services:

-
- **Auth Service (JWT-based)** – for authentication and authorization.
-
- **Optional Cache Layer (Redis)** – for high-traffic optimization (e.g., product listings).
-
- **CDN** – serves static assets and product images quickly worldwide.
-
- **Deployment Services** – e.g., Vercel/Netlify for frontend, Render/DigitalOcean for backend, MongoDB Atlas for DB.
-

---

2. Component Responsibilities

Component	Responsibilities
React SPA	- Displays product catalog, product detail, and cart pages.- Manages client-side routing via React Router.- Handles UI state (cart items, user session, filters).- Communicates with backend REST APIs using Axios or Fetch.- Stores JWT in memory or localStorage for API authentication.
Node.js + Express API	- Acts as a stateless REST API gateway.- Handles user registration, login, product fetch, cart management, and order creation.- Implements validation, error handling, and logging.- Issues and verifies JWTs for user authentication.- Interfaces with MongoDB through Mongoose ODM.- Optionally caches frequent queries (e.g., top products) using Redis.
MongoDB (Atlas / self-hosted)	- Stores user, product, category, cart, and order data.- Ensures high flexibility with a schema-less design (via Mongoose schema enforcement).- Uses indexes for fast lookups on product names, categories, and user IDs.
Auth Service (JWT)	- Issues signed JWT tokens at login.- Validates tokens on each protected route.- Decodes tokens to authorize user actions.- Can optionally issue refresh tokens.
Cache Layer (Redis, optional)	- Caches frequently requested endpoints (product lists, popular searches).- Reduces MongoDB load.- Invalidated when underlying data changes.
CDN (Cloudflare, Vercel, etc.)	- Delivers frontend static assets (JS, CSS, images).- Stores product images for low-latency global access.
Deployment Environment	- Frontend: Hosted on Vercel or Netlify.- Backend: Hosted on Render, Railway, or DigitalOcean App Platform.- Database: MongoDB Atlas (free or paid cluster).- Optional single-container deployment using Docker Compose for local dev.

3. Data Flow Description

1.
2. **User Interaction (Frontend)**
    -
    - User browses products → React sends `GET /api/products` to backend.
    -
    - React stores UI state locally (Context API or Redux).
    -
3.
4. **Backend Processing (Express)**
    -
    - Express receives request → passes to controller (e.g., ProductController).
    -
    - Controller queries MongoDB (optionally via Redis cache).
    -
    - Backend returns JSON response to frontend.
    -
5.
6. **Authentication Flow**
    -
    - User registers or logs in → `POST /api/users/login`.
    -
    - Server validates credentials → issues JWT.
    -
    - Frontend stores JWT → adds to Authorization headers for future requests.
    -
7.
8. **Cart Management**
    -
    - Guest cart stored in localStorage; for logged-in users, stored in MongoDB.
    -
    - On login, backend merges guest and user carts.
    -
9.
10. **Checkout Flow**
    -
    - Frontend sends `POST /api/orders` with order summary (placeholder).
    -
    - Backend stores order document → returns confirmation response.
    -
11.

12. **Static Asset Delivery**
    -
    - React static files (JS/CSS) served via CDN/Vercel edge.
    -
    - Product images fetched from CDN/Cloudinary bucket.
    -
13.

---

4. Deployment Architecture

Layer	Example Service	Description
Frontend	Vercel / Netlify	React app built and deployed to CDN edge nodes.
Backend	Render / Railway / DigitalOcean	Node.js app container exposing REST API endpoints.
Database	MongoDB Atlas	Cloud-hosted DB cluster with automatic backups and scaling.
Cache	Redis Cloud (optional)	For high-frequency product caching.
CI/CD	GitHub Actions	Automates lint, test, and deploy.

---

5. Design Trade-offs

🗄️ MongoDB

Pros:

-
- Flexible schema (good for evolving product catalog).
-
- JSON-like documents map naturally to JavaScript objects.
-
- Free tier on Atlas for quick development.
  Cons:
-
- Limited support for complex multi-document transactions.
-
- Denormalization can cause data duplication.
-
- May need careful index tuning for performance.
-

Alternative:

PostgreSQL for strict schema and ACID compliance if future payments or transactions require consistency.

🔐 JWT vs. Session Cookies

Aspect	JWT	Session Cookies
Scalability	Stateless; easy horizontal scaling (no session store).	Stateful; requires shared session store or sticky sessions.
Storage	Stored in localStorage or memory; easy API use.	Stored in cookies; CSRF protected.
Security Risks	Token theft if stored in localStorage.	CSRF if not protected.
Best For	APIs and mobile apps (stateless).	Traditional web apps with server rendering.

Decision: Use **JWT** (stored in memory or secure cookie) for stateless scaling and simple API integration.

⚛️ Client-side State: Context API vs. Redux

Feature	Context API	Redux Toolkit
Simplicity	Simple, built-in, minimal boilerplate.	More setup, but predictable state flow.
Performance	Good for small apps; re-renders on many updates.	Optimized for large global state.
DevTools	Basic debugging.	Advanced Redux DevTools.
Use Case	MVP (cart, auth, UI theme).	Large-scale (multi-module state, async logic).

Decision:

Use **React Context + useReducer** for MVP (cart and auth).

Consider migrating to Redux Toolkit once app grows.

---

6. Mermaid Diagram (Paste into docs)

```
flowchart TD
  subgraph Client
    A[React SPA] -->|HTTP Requests (Fetch/Axios)| B[Node.js Express API]
    A -->|Static Assets (CDN)| C[CDN / Vercel Edge]
  end

  subgraph Server
    B --> D[(MongoDB Atlas)]
    B -->|JWT Auth| E[Auth Service]
    B -->|Cache Lookup| F[(Redis Cache)]
  end

  subgraph Deployment
    C -->|Static Deployment| V[Vercel/Netlify]
    B -->|API Deployment| R[Render/DigitalOcean]
    D -->|Cloud Database| M[MongoDB Atlas]
  end

  A -.->|JWT Token Stored| A
  B -->|API Responses JSON| A
  D -->|Data (Products, Users, Orders)| B
  F -.optional caching.-> B
```

7. Summary

Architecture Type: Modular three-tier micro-frontend + REST API.

Core Principles: Stateless backend, scalable layers, component separation.

Key Benefits:

- Easy independent scaling of frontend/backend.
- Clear separation of concerns (UI, API, data).
- Cloud-friendly and CI/CD ready.
