E-commerce App

A full-stack e-commerce application built with Next.js (App Router), Express, MongoDB, and Docker Compose. This project demonstrates real production patterns including authentication, cart and checkout flows, admin workflows, SSR considerations, and containerized deployment.

Features

User

- Register / Login (cookie-based auth)
- Browse products
- Add to cart
- Checkout (mock payment)
- View orders

Admin

- Create and edit products
- View all orders
- Update order status

Engineering

- Next.js App Router with Server and Client Components
- Express REST API
- MongoDB with Mongoose
- Dockerized frontend, backend, and database
- Health check endpoint
- Production-ready environment handling

Tech Stack

- Frontend: Next.js 16 (App Router), React
- Backend: Node.js, Express
- Database: MongoDB
- Auth: JWT using HTTP-only cookies
- Containerization: Docker and Docker Compose

Project Structure

```
ecommerce-app/
  backend/
    src/
    Dockerfile
    package.json
```

```
frontend/
  app/
  Dockerfile
  package.json
docker-compose.yml
README.md
```

Environment Variables

Backend (Docker)

```
MONGO_URI=mongodb://mongo:27017/ecommerce
JWT_SECRET=dev_secret
FRONTEND_ORIGIN=http://localhost:3000
PORT=4000
```

Frontend

```
NEXT_PUBLIC_API_URL=http://localhost:4000
```

Note: NEXT_PUBLIC variables are exposed to the browser. Server-only values remain private.

Running Locally with Docker

Prerequisite: Docker Desktop on Mac, Windows, or Linux.

From the project root, run:
docker compose up --build

Frontend: http://localhost:3000
Backend health check: http://localhost:4000/health

Key Engineering Learnings

- Containers communicate over Docker networks, not localhost
- Next.js Server Components require server-safe environment variables
- Backend services must bind to 0.0.0.0 inside containers
- useSearchParams in the App Router requires a Suspense boundary
- Logs, health endpoints, and rebuilds are essential debugging tools

Why This Project Matters

This project mirrors real production issues including environment mismatches, SSR build failures, container networking bugs, and service orchestration challenges. It was built and debugged end to end, not scaffolded.

Next Steps

- Public deployment on Render, Railway, or Fly.io

- CI setup with GitHub Actions
- Stripe payment integration
- Role-based UI permissions