■ Project Recap: Full-Stack E-Commerce with Extension

■ Stack Overview

1. Frontend: React + React Router + Tailwind CSS → Product pages, cart, checkout, UI
2. Backend: Node.js + Express.js → API for products, cart, orders, users
3. Auth: JWT + bcrypt → Login, signup, route protection
4. Database: PostgreSQL (with pg or Prisma/Sequelize) → Store users, products, orders
5. Microservice: Java + Spring Boot → Generate invoices (PDF) for orders
6. Deployment: Vercel (frontend), Railway/Render/AWS (backend & Java)
7. Dev Tools: Git/GitHub, Postman, ESLint/Prettier

■ What You Need to Learn (Knowledge Points)

■■■ *Frontend (React + Tailwind)*

1. React basics (■ you know: useState, useEffect, etc.)
2. Routing with react-router-dom
3. Forms & controlled components
4. Auth flows (login/logout, store JWT in localStorage/context)
5. API calls with axios
6. Tailwind CSS basics: responsive classes, flexbox/grid, reusable components

■ *Backend (Node.js + Express)*

1. Express basics (express.Router, app.use)
2. Middlewares (cors, body-parser, custom auth middleware)
3. Password hashing with bcrypt
4. Token auth with JWT
5. Email sending with nodemailer
6. File uploads (multer, optional for product images)

■ *Database (PostgreSQL)*

1. Connect Node.js to Postgres with pg (or Prisma ORM)
2. Define tables: users, products, orders, order_items
3. CRUD queries (insert, select, update, delete)
4. Relations (user → orders, order → items, etc.)

■ *Microservice (Java + Spring Boot)*

1. Spring Boot basics (@RestController, @GetMapping, @PostMapping)
2. Dependency Injection (@Service, @Autowired)
3. Spring Data JPA (if connected to DB)
4. Generate PDFs with iText or PDFBox
5. Run microservice separately and call it from Node.js backend

■ *Security*

1. HTTPS setup (Let’s Encrypt, Vercel/Netlify auto SSL)
2. Always hash passwords
3. Never log/store raw sensitive data
4. Use environment variables (.env)

■ Project Milestones (Step by Step)

1. Setup: Init repo, create /client (React) and /server (Node.js), setup PostgreSQL
2. Frontend: Homepage, product listing, product detail, cart, login/register pages
3. Backend: Setup API, CRUD for products, Auth with JWT, Cart & Orders API
4. Frontend-Backend Connection: axios calls, JWT login, cart sync
5. Database Integration: Connect Node.js to PostgreSQL, implement schema
6. Microservice: Java Spring Boot → /invoice/{orderId} generates PDF invoice
7. Deployment: Vercel (frontend), Railway/Render (backend), deploy Java microservice
8. Optional Enhancements: Admin dashboard, payments (Stripe/PayPal), Next.js migration

■ Learning Strategy

1. Start simple: React UI → Express API → DB → Auth
2. Add complexity gradually (cart, orders, invoicing)
3. Don’t worry about SEO/SSR yet, save for senior-level project
4. Reuse Java microservice later when upgrading project