PHASE 1: Backend Setup (Node.js + Prisma + Supabase PG)

1. Initialize Project

- Create a new backend folder: inventory-backend
- Init project with `npm init -y`
- Install core packages: express, typescript, ts-node-dev, dotenv, etc.
- Setup folder structure:

```
src/
  domain/
  application/
  infrastructure/
  routes/
  config/
```

2. TypeScript + Configs

- Add tsconfig.json with module/paths setup
- Setup nodemon.json or ts-node-dev for hot reload
- Create .env for config

3. Prisma + Supabase

- Install prisma & @prisma/client
- Run npx prisma init
- Set Supabase PostgreSQL URL in .env
- Define schema (inventory, images, etc.) in schema.prisma
- Run npx prisma migrate dev --name init
- Add Prisma service layer inside infrastructure/

4. Core Clean Architecture

- Setup interfaces/models in domain/
- Implement business logic in application/ (CRUD + stock check use cases)
- Build database adapters using Prisma in infrastructure/
- Define route handlers in routes/
- Wire everything up in main.ts or index.ts

5. Auth (Optional)

- Decide if you'll use Supabase Auth or custom JWT
- Setup middleware for protected routes (if needed)

6. Testing

- Test all routes using Postman / Thunder Client
- Add basic unit tests (optional at this phase)

PHASE 2: Dockerize the Backend

1. Docker Setup

- Create Dockerfile for Node backend
- Create .dockerignore
- Create docker-compose.yml with:
  - Node backend container
  - Optional local PostgreSQL container (if testing locally)
- Configure service to connect to either:
  - Local Postgres (in dev)
  - Supabase (in staging/prod)

PHASE 3: Frontend Setup (React + Zustand)

1. React Init

- Create new frontend project: inventory-frontend
- Init with TypeScript
- Install Tailwind CSS
- Setup folder structure for components, pages, hooks, etc.

2. Zustand Store

- Create inventory store with Zustand
- Functions: fetchItems, addItem, updateItem, deleteItem

3. UI Pages

- Inventory list page
- Item details / edit page
- Create item form with validation (React Hook Form + Zod/Yup)
- Image upload field (for Supabase Storage)

4. API Integration

- Connect to your backend using Axios/Fetch
- Use optimistic UI updates with Zustand
- Handle errors & loading states

5. Deployment

- Deploy backend to Render / Railway / Fly.io (from Docker container)
- Deploy frontend to Netlify or Vercel

- Add .env files for API URLs and storage endpoints

PHASE 4: React Native Setup (Later)

- Init with Expo
- Setup Zustand (reusable store logic)
- Connect to same backend endpoints
- Build minimal mobile version

Final Cleanups

- Add README with setup instructions
- Add .env.example files
- Setup Prettier, ESLint, and commit hooks