The Petal Pouches
Development Workflow & Architecture

Comprehensive step-by-step guide — Backend-first development with Supabase + Cloudinary, Vite + React + Tailwind frontend, Shiprocket shipping integration.

Table of Contents

1. Project Overview

The Petal Pouches is an Instagram-first gifting e-commerce platform targeting teenage and young girls. This document guides a backend-first development approach using Supabase (PostgreSQL) for data, Cloudinary for media, Vite + React + Tailwind for frontend, and serverless/Node backend for integrations (payments, shipping). The goal: a performant, 3D-capable, highly engaging website with scalable architecture.

2. Development Philosophy & Strategy (Backend-first)

Recommended strategy: Build a minimal secure backend first (data model, core APIs, auth, payment and shipping integrations), then scaffold the frontend to consume those APIs, and finally enhance the UI/UX including 3D experiences and progressive optimizations.

Why backend-first? Benefits:

- Provides stable API contracts for frontend to consume (reduces rework).
- Enables early testing of end-to-end flows (orders, payments, shipments).
- Secures secrets and sensitive logic server-side (payment keys, Shiprocket credentials).
- Allows seeding of data so frontend development uses real data.

3. Prerequisites & Accounts

Before you start, create accounts and obtain keys for these services:

Service	Purpose / Notes
GitHub	Repository hosting
Vercel	Frontend hosting (or Netlify)
Render / Railway / Heroku	Backend Node host (optional)
Supabase	PostgreSQL database + Auth + Storage (we will use for structured data)
Cloudinary	Image & media hosting + optimization (store image URLs in DB)
Shiprocket	Shipping aggregator API for rates & tracking (India)
Razorpay or Stripe	Payment gateway (Razorpay recommended for India)
Google Analytics / Plausible	Analytics
Sentry (optional)	Error monitoring

Create these early and save keys in a password manager (1Password, Bitwarden). You will add them later as environment variables.

4. Step-by-step Development Workflow (High level)

Phase 0 — Preparation (1 week)

1. Create GitHub repo and initialize README, license, .gitignore.
2. Create Supabase project and configure DB API keys.
3. Create Cloudinary account and note API key/config.
4. Decide hosting: Vercel for frontend, Render/Heroku for backend (or use Vercel serverless functions).

Phase 1 — Backend Core (2–3 weeks)

5. Design and implement DB schema in Supabase (SQL).
6. Create seed scripts and upload sample product images to Cloudinary.
7. Implement core REST API endpoints using Node.js + Express or serverless functions (see API Contracts).
8. Implement Supabase Auth or JWT-based authentication.
9. Integrate payment provider in test mode (Razorpay/Stripe).
10. Integrate Shiprocket API for quote & shipment creation (test sandbox).

Phase 2 — Frontend Skeleton (2–3 weeks)

11. Initialize Vite React project with Tailwind CSS and TypeScript.
12. Create basic pages: Home, Shop, Product Detail, Cart, Checkout, Track.
13. Wire pages to backend APIs for products and checkout flows.
14. Implement client-side cart state and localStorage persistence.

Phase 3 — Personalization & 3D (3–6 weeks)

15. Add personalization preview & gift builder UI.
16. Prototype 3D hero in Spline and embed; later migrate to react-three-fiber for deeper control.
17. Optimize media delivery: move large assets to Cloudinary and serve via CDN.

Phase 4 — Polish & Launch (2–4 weeks)

18. Add analytics, error tracking, performance budget & monitoring.
19. Complete QA, accessibility testing (WCAG), and SEO optimizations.
20. Prepare deployment pipelines, test payments in production, and launch.

5. Exact File-by-File, Order-to-Create (Backend-first)

Start by creating the backend project folder and files. Create, test, and then move to the next file. This ensures a working API for frontend to consume.

Backend-first File Creation Order (priority 1 → 12):

21. **1. package.json —** Initialize Node.js project. Add scripts: start, dev, seed, lint.
22. **2. .env.example —** Define environment variables (see list below).
23. **3. src/server.js or src/index.js —** Express server entrypoint or serverless handler.
24. **4. src/config/supabaseClient.js —** Server-side Supabase client using service role key.
25. **5. src/config/cloudinary.js —** Cloudinary config helper for uploads.
26. **6. src/db/migrations/001_create_tables.sql —** SQL to create products, users, orders, shipments, gift_previews.
27. **7. scripts/seed_db.js —** Seed DB with sample products and Cloudinary URLs.
28. **8. src/routes/products.js —** GET /products, GET /products/:id, POST /products (admin).
29. **9. src/routes/auth.js —** Auth routes if using custom auth (signup/login) — optional if using Supabase Auth.
30. **10. src/routes/orders.js —** POST /create-order, GET /orders/:id.
31. **11. src/routes/shipping.js —** POST /quote-shipping, POST /create-shipment, GET /track/:awb.

32. **12. src/routes/payments.js —** Webhook endpoint for payment provider; route to verify & update orders.

Environment variables to add to .env (server-side only):

- SUPABASE_URL
- SUPABASE_SERVICE_ROLE_KEY
- CLOUDINARY_CLOUD_NAME
- CLOUDINARY_API_KEY
- CLOUDINARY_API_SECRET
- SHIPROCKET_API_KEY (if applicable)
- RAZORPAY_KEY_ID / STRIPE_SECRET_KEY
- WEBHOOK_SECRET
- NODE_ENV

6. Frontend Tasks & File Order (Enhance UI later)

Frontend can be developed once core backend endpoints are available. Start with a minimal UI that consumes APIs, then progressively enhance visuals and 3D.

33. **1. Initialize Vite Project —** vite + react + typescript template. Install Tailwind CSS, ESLint, Prettier.
34. **2. src/main.tsx —** App entrypoint; set up global providers (Auth, Router).
35. **3. src/lib/api.ts —** Wrapper around fetch/axios to call backend APIs; include error handling.
36. **4. src/pages/Home.tsx —** Home page skeleton (fetch featured products).
37. **5. src/pages/Shop.tsx —** Product listing page with filters.
38. **6. src/pages/Product/[id].tsx —** Product Detail Page (PDP) with personalization UI and image gallery using Cloudinary URLs).
39. **7. src/pages/Cart.tsx —** Cart UI and local persistence (localStorage).
40. **8. src/pages/Checkout.tsx —** Checkout form: address, pincode, shipping quote, payment trigger.
41. **9. src/components/3d/HeroCanvas.tsx —** Add Spline iframe first, then convert to react-three-fiber as enhancement.
42. **10. src/components/ui/* —** Buttons, Inputs, Modals, Toasts components for consistent design.
43. **11. src/hooks/useCart.ts —** Cart management with context or Zustand.
44. **12. src/lib/cloudinary.ts —** Helper for signed uploads from admin panel (if enabling direct client uploads).

7. API Contracts (Endpoints & Payloads)

Below are the primary API endpoints and example request/response payloads. Keep these stable; the frontend will consume them.

Endpoint	Method	Description / Payload
/api/products	GET	Query params: page, limit, q (search). Returns: { products: [], meta: {} }
/api/products/:id	GET	Returns detailed product with variants and image URLs.
/api/quote-shipping	POST	Body: { dest_pincode, weight_g }. Returns: [{ courier, price, est_days }]

/api/create-order	POST	Body: { user_id, items: [{product_id, variant_id, qty, personalization}], shipping_address }. Returns: { order_id, payment_token }
/api/webhook/payment	POST	Payment provider webhook — validate signature, update order status.
/api/create-shipment	POST	Body: { order_id }. Creates shipment with Shiprocket, saves AWB.
/api/track/:awb	GET	Returns tracking status and history.

8. Supabase Database Schema (SQL & Explanations)

Use PostgreSQL on Supabase. Below are the core tables and an example CREATE TABLE DDL for each. Run them in Supabase SQL editor or your migration tool.

Note: UUIDs, timestamps, and JSONB are used for flexibility and compatibility with personalization.

1) Table: users

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Unique ID
name	text	—	NOT NULL	User name
email	text	—	UNIQUE	Login email
password	text	—	NOT NULL	Hashed
created_at	timestamptz	now()	—	Registered at

2) Table: addresses

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Unique ID
user_id	uuid	—	FK → users.id	Owner
line1	text	—	NOT NULL	Address line
city	text	—	NOT NULL	City
state	text	—	NOT NULL	State
zip_code	text	—	NOT NULL	Postal code

3) Table: products

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Product ID
title	text	—	NOT NULL	Product name
price	int4	—	NOT NULL	Base price
sku	text	—	UNIQUE	Product code
category	text	—	—	Category
created_at	timestamptz	now()	—	Created

4) Table: product_variants

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Variant ID
product_id	uuid	—	FK → products.id	Parent
sku	text	—	UNIQUE	Variant code
attributes	jsonb	—	—	Color/size
inventory_count	int4	0	—	Stock

5) Table: orders

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Order ID
user_id	uuid	—	FK → users.id	Buyer
status	text	'pending'	—	Order status
total	int4	—	—	Total amount
payment_status	text	'unpaid'	—	Payment state

6) Table: order_items

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Item ID
order_id	uuid	—	FK → orders.id	Order
product_variant_id	uuid	—	FK → product_variants.id	Variant
qty	int4	—	NOT NULL	Quantity
price_at_purchase	int4	—	—	Snapshot price

7) Table: order_discounts

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Event ID
order_id	uuid	—	FK → orders.id	Order
coupon_id	uuid	—	FK → coupons.id	Coupon
discount_applied	int4	0	NOT NULL	Discount value
created_at	timestamptz	now()	—	Created

8) Table: shipments

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Shipment ID
order_id	uuid	—	FK → orders.id	Order
courier	text	—	—	Courier name
awb	text	—	UNIQUE	Tracking ID
status	text	—	—	Delivery state

9) Table: gift_previews

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Preview ID
user_id	uuid	—	FK → users.id	Owner
preview_url	text	—	—	Image URL
personalization	jsonb	—	—	Custom text

10) Table: categories

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Category ID
name	text	—	UNIQUE	Category name
description	text	—	—	Details

11) Table: coupons

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Coupon ID
code	text	—	UNIQUE	Code
discount_type	text	'flat'	—	flat/percent
discount_value	int4	—	NOT NULL	Value
is_active	boolean	true	—	Active

12) Table: applied_coupons

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Record ID
order_id	uuid	—	FK → orders.id	Order
coupon_id	uuid	—	FK → coupons.id	Coupon
discount_amount	int4	—	—	Discount (₹)

13) Table: reviews

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Review ID
product_id	uuid	—	FK → products.id	Product
user_id	uuid	—	FK → users.id	User
rating	int2	—	NOT NULL	Rating (1–5)

14) Table: wishlist

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Wishlist ID
user_id	uuid	—	FK → users.id	User
product_id	uuid	—	FK → products.id	Product

15) Table: cart_items

Column	Type	Default	Constraints	Notes
id	uuid	gen_random_uuid()	Primary key	Cart ID
user_id	uuid	—	FK → users.id	User
product_variant_id	uuid	—	FK → product_variants.id	Variant
qty	int4	—	NOT NULL	Qty

9. Cloudinary Integration Workflow

Cloudinary will host images and deliver optimized variants. We store the returned secure URLs in Supabase product records.

45. 1. Create Cloudinary account and get CLOUD_NAME, API_KEY, API_SECRET.
46. 2. For admin uploads, use signed uploads: create a signed upload endpoint on the backend (server signs upload preset).
47. 3. Admin (browser) uploads image to Cloudinary via direct POST to Cloudinary upload endpoint (or via the signed URL).
48. 4. Cloudinary returns secure URL and public_id; save URL in products.images array in Supabase.

49. 5. Frontend uses Cloudinary URLs directly. Use transformation params for thumbnails, progressive delivery, and responsive sizes.

Signed upload sample (backend pseudocode):

POST /api/admin/sign-cloudinary
Backend: generate signature using CLOUDINARY_API_SECRET and return signature, timestamp, and api_key to client.
Client: upload file directly to https://api.cloudinary.com/v1_1/<cloud_name>/image/upload with signature.

10. Shipping & Payments Integration

Payments (Razorpay/Stripe): Use test mode during development. Create order on server, create payment intent, return token to client, verify webhook to update order, then create shipment.

Shipping (Shiprocket): Steps for integration:

- Use Shiprocket API to fetch courier options and delivery estimates by pincode and weight.
- Show options on checkout; user selects courier/service.
- After payment success, create shipment via Shiprocket and save AWB in shipments table.
- Expose /track/:awb endpoint to show live tracking status by querying Shiprocket or courier API.

11. Deployment & Environment

Recommended hosting:

Component	Hosting
Frontend	Vercel (recommended) — connect GitHub repo, automatic deploys
Backend (API)	Render / Railway / Vercel Serverless Functions — ensure env variables are set
Database	Supabase (managed)
Media	Cloudinary

Environment variable checklist (again):

- SUPABASE_URL
- SUPABASE_SERVICE_ROLE_KEY
- CLOUDINARY_CLOUD_NAME
- CLOUDINARY_API_KEY
- CLOUDINARY_API_SECRET
- SHIPROCKET_API_KEY (if applicable)
- RAZORPAY_KEY_ID / STRIPE_SECRET_KEY
- WEBHOOK_SECRET
- NODE_ENV

12. Testing, Monitoring & QA Checklist

- Unit tests for critical backend functions (payment verification, shipping creation).
- Integration tests for create-order → payment → webhook → create-shipment flow (use test keys).
- Manual QA on low-end devices for 3D and mobile fallback.

- Accessibility audit (Lighthouse, axe).
- Performance audit (Lighthouse performance score, reduce JS bundle size).
- Error monitoring (Sentry) and logging for server endpoints.

13. Final Detailed Directory Structure

Below is the complete recommended repo layout for the project. Use this as the canonical structure for development.

```
the-petal-pouches/
├─ .github/
│  └─ workflows/
│     └─ ci.yml
├─ backend/
│  ├─ package.json
│  ├─ .env.example
│  ├─ src/
│  │  ├─ index.js            # Express / serverless entry
│  │  ├─ config/
│  │  │  ├─ supabaseClient.js
│  │  │  └─ cloudinary.js
│  │  ├─ routes/
│  │  │  ├─ products.js
│  │  │  ├─ orders.js
│  │  │  ├─ shipping.js
│  │  │  └─ payments.js
│  │  ├─ lib/
│  │  │  ├─ shiprocket.js
│  │  │  └─ paymentsHelper.js
│  │  └─ migrations/
│  │     └─ 001_create_tables.sql
│  └─ scripts/
│     └─ seed_db.js
├─ frontend/
│  ├─ package.json
│  ├─ vite.config.ts
│  ├─ tsconfig.json
│  ├─ public/
│  │  ├─ favicon.ico
│  │  └─ placeholder.png
│  ├─ src/
│  │  ├─ main.tsx
│  │  ├─ App.tsx
│  │  ├─ lib/
│  │  │  ├─ api.ts
│  │  │  └─ cloudinary.ts
```

```
|  |  ├─ pages/
|  |  |  ├─ Home.tsx
|  |  |  ├─ Shop.tsx
|  |  |  ├─ product/
|  |  |  |  └─ [id].tsx
|  |  |  ├─ Cart.tsx
|  |  |  ├─ Checkout.tsx
|  |  |  └─ Track.tsx
|  |  ├─ components/
|  |  |  ├─ ui/
|  |  |  |  ├─ Button.tsx
|  |  |  |  └─ Modal.tsx
|  |  |  ├─ product/
|  |  |  |  └─ ProductCard.tsx
|  |  |  └─ 3d/
|  |  |     ├─ HeroCanvas.tsx
|  |  |     └─ Product3DViewer.tsx
|  |  ├─ hooks/
|  |  |  ├─ useCart.ts
|  |  |  └─ useDevice.ts
|  |  ├─ styles/
|  |  |  ├─ globals.css
|  |  |  └─ tailwind.css
|  |  └─ data/
|  |     └─ seed-products.json
├─ docs/
|  └─ The_Petal_Pouches_Dev_Workflow.docx
└─ README.md
```

14. Appendix: Useful Commands & Templates

Frontend: init vite + packages:

```
npm create vite@latest frontend -- --template react-ts
cd frontend
npm install
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

Backend: init node:

```
mkdir backend && cd backend
npm init -y
npm i express dotenv axios pg @supabase/supabase-js cloudinary
```

Seed script run:

```
node backend/scripts/seed_db.js
```

Run frontend dev server:

```
cd frontend
npm run dev
```

Run backend dev server:

```
cd backend
npm run dev
```