**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

2. Development Philosophy & Strategy (Backend-first)

3. Prerequisites & Accounts

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

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

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

7. API Contracts (Endpoints & Payloads)

8. Supabase Database Schema (SQL & explanations)

9. Cloudinary Integration Workflow

10. Shipping & Payments Integration

11. Deployment & Environment

12. Testing, Monitoring & QA Checklist

13. Final Detailed Directory Structure

14. Appendix: Useful Commands & Templates

## 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)

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

#### Phase 2 — Frontend Skeleton (2–3 weeks)

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

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

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

#### Phase 4 — Polish & Launch (2–4 weeks)

1. Add analytics, error tracking, performance budget & monitoring.
2. Complete QA, accessibility testing (WCAG), and SEO optimizations.
3. 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):

1. **1. package.json —** Initialize Node.js project. Add scripts: start, dev, seed, lint.
2. **2. .env.example —** Define environment variables (see list below).
3. **3. src/server.js or src/index.js —** Express server entrypoint or serverless handler.
4. **4. src/config/supabaseClient.js —** Server-side Supabase client using service role key.
5. **5. src/config/cloudinary.js —** Cloudinary config helper for uploads.
6. **6. src/db/migrations/001\_create\_tables.sql —** SQL to create products, users, orders, shipments, gift\_previews.
7. **7. scripts/seed\_db.js —** Seed DB with sample products and Cloudinary URLs.
8. **8. src/routes/products.js —** GET /products, GET /products/:id, POST /products (admin).
9. **9. src/routes/auth.js —** Auth routes if using custom auth (signup/login) — optional if using Supabase Auth.
10. **10. src/routes/orders.js —** POST /create-order, GET /orders/:id.
11. **11. src/routes/shipping.js —** POST /quote-shipping, POST /create-shipment, GET /track/:awb.
12. **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.

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

## 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 |

## 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 |

## 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 |

## 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 |

## 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 |

## 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 |

## 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 |

## 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 |

## 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 |

## Table: categories

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column** | **Type** | **Default** | **Constraints** | **Notes** |
| id | uuid | gen\_random\_uuid() | Primary key | Category ID |
| name | text | — | UNIQUE | Category name |
| description | text | — | — | Details |

## 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 |

## 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 (₹) |

## 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) |

## 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 |

## 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.

1. 1. Create Cloudinary account and get CLOUD\_NAME, API\_KEY, API\_SECRET.
2. 2. For admin uploads, use signed uploads: create a signed upload endpoint on the backend (server signs upload preset).
3. 3. Admin (browser) uploads image to Cloudinary via direct POST to Cloudinary upload endpoint (or via the signed URL).
4. 4. Cloudinary returns secure URL and public\_id; save URL in products.images array in Supabase.
5. 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