NOVA Fashion Store

Backend Integration Guide

---

1. Why Add a Backend?

The current store works entirely client-side. Orders go via EmailJS (one email per order), stock is hardcoded in products.js, and there is no admin panel. A backend gives you:

- A real orders database — searchable, filterable, exportable
- Live inventory management — update stock without touching code
- An admin dashboard — see orders, mark them fulfilled, track revenue
- Newsletter email list — store subscriber emails from the popup
- Product reviews from real customers — stored and retrieved dynamically
- Optional: customer accounts, order tracking pages

2. Recommended Stack

Since NOVA is already on Vercel, the cleanest path is to stay in the Vercel ecosystem. This requires zero new accounts or infrastructure.

Layer	Tool	Cost	Why
API Routes	Vercel Serverless Functions	Free	Already on Vercel, no new server
Database	Supabase (PostgreSQL)	Free up to 500MB	Instant dashboard, REST + realtime
Auth (admin)	Supabase Auth	Free	Built-in, email/password
Images	Cloudinary	Free 25GB	Transform, CDN, fast
Email	Resend (or keep EmailJS)	Free 3k/mo	Reliable transactional email

3. New Folder Structure

You keep all your existing frontend files exactly as they are. You add an /api folder:

```
nova-store/
├── index.html          (unchanged)
```

```
├── styles.css          (unchanged)
├── products.js         (will become dynamic — fetched from DB)
├── ...all other .js files (unchanged)
│
├── api/                ← NEW — Vercel serverless functions
│   ├── orders.js       ← POST: create order  GET: list orders (admin)
│   ├── products.js     ← GET: fetch products  PATCH: update stock
│   ├── newsletter.js   ← POST: save subscriber email
│   └── reviews.js      ← GET: fetch reviews  POST: submit review
│
├── admin/              ← NEW — simple admin dashboard
│   └── index.html      ← protected, shows orders & stock
│
└── vercel.json         ← routing config
```

4. Step-by-Step Integration Plan

Phase 1 — Database Setup (30 min)

1. Go to supabase.com → Create a free project
2. In Table Editor, create these tables:

```
orders
  id           uuid primary key default gen_random_uuid()
  order_number text not null unique
  customer_name    text
  customer_address text
  customer_phone   text
  items        jsonb    -- array of cart items
  total        numeric
  status       text default 'pending'
  created_at   timestamptz default now()

products
  id           text primary key  -- matches your existing product IDs
  stock        integer
  size_stock   jsonb
  price        numeric
  old_price    numeric
  active       boolean default true

newsletter_subscribers
  id    uuid primary key default gen_random_uuid()
```

```
  email text unique not null
  lang  text default 'en'
  created_at timestamptz default now()

reviews
  id         uuid primary key default gen_random_uuid()
  product_id text references products(id)
  stars      integer check (stars between 1 and 5)
  text_en    text
  author     text
  verified   boolean default false
  created_at timestamptz default now()
```

Phase 2 — API Routes (2-3 hours)

Create a /api folder in your repo with one file per resource. Each is a standard Vercel serverless function (Node.js).

api/orders.js

```
// POST /api/orders — create new order
// GET  /api/orders — list orders (admin, requires secret key)

import { createClient } from '@supabase/supabase-js';

const supabase = createClient(
  process.env.SUPABASE_URL,
  process.env.SUPABASE_SERVICE_KEY
);

export default async function handler(req, res) {
  if (req.method === 'POST') {
    const { name, address, phone, cart, total } = req.body;
    const orderNumber = 'NOVA-' + Math.floor(100000 + Math.random() *
900000);
    const { data, error } = await supabase
      .from('orders')
      .insert({ order_number: orderNumber, customer_name: name,
                customer_address: address, customer_phone: phone,
                items: cart, total });
    if (error) return res.status(500).json({ error: error.message });
    return res.json({ success: true, orderId: orderNumber });
  }
  // GET — admin only (check x-admin-key header)
```

```
  ...
}
```

api/products.js

```
// GET  /api/products — returns all products with live stock
// PATCH /api/products/:id — update stock (admin)

export default async function handler(req, res) {
  if (req.method === 'GET') {
    const { data } = await supabase.from('products').select('*');
    return res.json(data);
  }
  if (req.method === 'PATCH') {
    const { id } = req.query;
    const { stock, sizeStock } = req.body;
    await supabase.from('products').update({ stock, size_stock: sizeStock
}).eq('id', id);
    return res.json({ success: true });
  }
}
```

Phase 3 — Connect Frontend to API (1-2 hours)

This is the only part where you change existing frontend files. The changes are small and isolated.

checkout.js — Replace EmailJS with API call

```
// BEFORE (EmailJS):
await emailjs.send(SERVICE_ID, TEMPLATE_ID, { ... });

// AFTER (your API):
const res = await fetch('/api/orders', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ name, address, phone, cart, total })
});
const data = await res.json();
if (!data.success) throw new Error(data.error);
```

index.html — Load products dynamically

```
// In init(), replace static products array usage:
async function loadProducts() {
  const res = await fetch('/api/products');
  window.products = await res.json();
  renderProducts();
}
// Call loadProducts() instead of renderProducts() in init()
```

engagement.js — Save newsletter subscribers

```
// In the newsletter form submit handler:
await fetch('/api/newsletter', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ email, lang: currentLang })
});
```

Phase 4 — Admin Dashboard (2-3 hours)

A simple password-protected HTML page at /admin/index.html that:

- Shows a table of all orders with status (pending / shipped / cancelled)
- Lets you click an order to see full details
- Lets you update stock numbers per product
- Shows total revenue and order count

Authentication is handled by a simple secret key in a header — no complex auth system needed for a solo operation.

Phase 5 — Environment Variables

In Vercel dashboard → your project → Settings → Environment Variables, add:

```
SUPABASE_URL          = https://xxxx.supabase.co
SUPABASE_SERVICE_KEY  = eyJ...  (from Supabase project settings)
ADMIN_SECRET_KEY      = any long random string you choose
```

These are never exposed to the browser. Your API routes read them with process.env.SUPABASE_URL.

5. What Files You Send Me — Session by Session

You never need to send all files at once. Here is what to share for each feature:

Session Goal	Send Me	I Build You
Orders API	checkout.js	api/orders.js + updated checkout.js
Live product stock	products.js	api/products.js + updated init code snippet
Newsletter storage	engagement.js	api/newsletter.js + updated handler snippet
Reviews system	reviews.js	api/reviews.js + updated reviews.js
Admin dashboard	Nothing (I build from scratch)	admin/index.html (standalone)
Supabase DB setup	Nothing	Full SQL schema to paste in Supabase editor
vercel.json config	Nothing	vercel.json with correct routing rules

6. Estimated Time Investment

Phase	Work	Time
Database setup	Create Supabase project + tables	30 min
Orders API	api/orders.js + update checkout.js	1-2 hrs
Products API	api/products.js + update init	1 hr
Newsletter API	api/newsletter.js + update engagement.js	30 min
Admin dashboard	admin/index.html	2-3 hrs
Reviews API	api/reviews.js + update reviews.js	1 hr
Testing + deploy	Verify all endpoints on Vercel	1 hr
TOTAL		~7-9 hrs

7. Package Dependencies

Install these in your project root before deploying API routes:

```
npm init -y              (if no package.json yet)
npm install @supabase/supabase-js
npm install resend       (if replacing EmailJS for email)
```

Vercel will automatically install node_modules on every deploy.
