NOVA

Complete Backend Build Guide

What to do outside Claude . Step-by-step per phase . Every message with exact files to share

How to use this PDF Each phase has two parts: (1) STEPS -- things you do yourself outside Claude, websites to visit, commands to run. (2) MESSAGES -- one block per message with the files to attach if new session + the copy-paste phrase.

PHASE 0

SETUP -- Accounts, Config Files, Node.js

~1 hour -- do this before any coding

PART A -- THINGS YOU DO YOURSELF (no coding)

Websites to visit, buttons to click

1. Create Supabase account and project

`Go to supabase.com`

1	**Sign up** Click "Start for free". Sign up with GitHub (fastest) or email.
2	**New project** After login -> click "New project" -> fill in: Name: nova-store Database password: click Generate -> SAVE this password somewhere safe Region: Europe West 1 (Ireland) -- closest free region to Tunisia Plan: Free -> click Create new project
3	**Wait** A progress bar appears. Wait 1-2 minutes until it says "Your project is ready".

4	**Get Project URL** Left sidebar -> gear icon (Settings) -> click "API" Copy the "Project URL" -> looks like: https://abcdefghijk.supabase.co
5	**Get service_role key** Same API page -> scroll to "Project API keys" You see two keys: anon (public) and service_role Click the eye icon next to service_role to reveal it -> copy the full string [!] Copy service_role NOT anon. They look similar but service_role has full DB access.

2. Create Upstash account and Redis database

`Go to upstash.com`

1	**Sign up** Click "Start for free" -> sign up with Google or email.
2	**Create Redis database** Click "Create database" -> Name: nova-ratelimit Type: Regional (not Global -- Regional is on the free plan) Region: eu-west-1 -> click Create
3	**Get REST URL** Once created, click the database name to open it. Scroll down to the "REST API" section (below the Connection section) Copy the UPSTASH_REDIS_REST_URL -> looks like: https://xxxx.upstash.io
4	**Get REST Token** Same REST API section -> copy the UPSTASH_REDIS_REST_TOKEN -> long string

3. Create Resend account and API key

`Go to resend.com`

1	**Sign up** Click "Get started" -> sign up with email -> confirm your email address when they send a verification email.
2	**Skip domain setup** Resend asks you to add a domain. Click "Skip" or "I'll do this later". You can add your real domain after you buy it. For now the test sender works fine.
3	**Create API key** Left sidebar -> "API Keys" -> click "Add API key" Name: nova-orders Permission: Full access -> click Add

4 **COPY THE KEY NOW**

The key is shown ONLY ONCE on screen. Copy it immediately. Format: re_xxxxxxxxxxxxxxxxxxxxxxxxxx If you miss it: delete the key and create a new one.

4. Add all 8 variables to Vercel

Go to vercel.com -> your nova-store project -> Settings tab -> Environment Variables

For each variable below: click "Add New" -> type Name -> paste Value -> check all 3 environments (Production, Preview, Development) -> Save. Repeat for all 8.

Variable Name	Value format	Where it comes from
SUPABASE_URL	https://xxxx.supabase.co	Supabase -> Settings -> API -> Project URL
SUPABASE_SERVICE_KEY	eyJhbGciOi... (long)	Supabase -> Settings -> API -> service_role key
UPSTASH_REDIS_REST_URL	https://xxxx.upstash.io	Upstash -> your DB -> REST API section
UPSTASH_REDIS_REST_TOKEN	AXxxxxxxxx... (long)	Upstash -> your DB -> REST API section
RESEND_API_KEY	re_xxxxxxxxxx	Resend -> API Keys -> shown once at creation
RESEND_TO_EMAIL	your@email.com	Your own email -- where you receive orders
ADMIN_SECRET	type 32+ random chars yourself	You invent this -- e.g. xK9mP2qRn5vT1wY4uI7a
JWT_SECRET	type 32+ different random chars	You invent this -- must differ from ADMIN_SECRET

After adding all 8 variables Go to Vercel -> Deployments tab -> click "..." on your latest deployment -> Redeploy. Variables only activate after redeploy.

5. Install Node.js on your PC

Go to nodejs.org

1 **Download LTS**

On the homepage click the left button "LTS" -> downloads a .msi installer file.

2 **Install**

Double click the .msi file -> Next -> Next -> Install -> Finish. All defaults are fine. Takes 1 min.

3 **Verify**

Open Command Prompt (Windows key -> type cmd -> Enter) Type: node --version -> should show v18 or higher Type: npm --version -> should show a number like 10.x.x

6. Create your local .env file

In your nova-store folder on your PC, create a file named exactly .env (no extension after the dot). Open it in Notepad and paste this, replacing each value with your real ones:

```
# NOVA local environment -- NEVER commit this to GitHub
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_SERVICE_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
UPSTASH_REDIS_REST_URL=https://your-db.upstash.io
UPSTASH_REDIS_REST_TOKEN=AXxxxxxxxxxxxxxxx
RESEND_API_KEY=re_xxxxxxxxxxxxxxx
RESEND_TO_EMAIL=your@email.com
ADMIN_SECRET=yourLongRandomStringHere32PlusChars
JWT_SECRET=anotherDifferentLongRandomStringHere
```

PART B -- MESSAGES TO SEND CLAUDE

Phase 0 has 3 messages -- one file group per message

MSG 0-1 database/schema.sql -- all 11 tables

Claude creates:

`database/schema.sql`

All CREATE TABLE statements for SQLite + Supabase (PostgreSQL compatible)

If new session -- attach:

No files needed

Copy-paste phrase:

```
NOVA Phase 0, Message 1.
# No files to share -- generate from scratch.
Please generate database/schema.sql with these 11 tables:
orders (id, order_number, customer_name, customer_address,
customer_phone, items JSON, total, status, lang, created_at)
products (id, name_en, name_ar, desc_en, desc_ar,
category_id, price, old_price, stock, active, created_at)
product_colors (id, product_id, name_en, name_ar, hex)
product_images (id, color_id, url, sort_order)
product_sizes (id, product_id, size_label, stock)
categories (id, name_en, name_ar, slug)
category_sizes (id, category_id, size_label)
newsletter_subscribers (id, email, lang, created_at)
reviews (id, product_id, stars, text_en, text_ar,
author, verified, created_at)
admin_users (id, email, password_hash, created_at)
rate_limit_log (id, ip, endpoint, created_at)
Use SQL compatible with both SQLite and PostgreSQL.
Add a comment above each table explaining what it stores.
```

MSG 0-2 .env.example + .gitignore

Claude creates:

`.env.example`

All 8 variable names with descriptions -- safe to commit to GitHub

`.gitignore`

Protects .env and database/nova.db from ever going to GitHub

If new session -- attach:

`database/schema.sql`

Helps Claude reference variable names correctly

Copy-paste phrase:

```
NOVA Phase 0, Message 2.
# Sharing: database/schema.sql
Please generate two files:
.env.example
Show all 8 variable names with a comment explaining each one.
No real values -- placeholder text only.
Safe to commit to GitHub so other developers know what is needed.
.gitignore
Must include: .env, database/nova.db, node_modules/,
.DS_Store, *.log, .vercel
Generate both in one reply.
```

MSG 0-3 vercel.json + package.json

Claude creates:

`vercel.json`

Routes /api/* and /admin/* correctly. Adds basic security headers.

`package.json`

npm dependencies and scripts for the project

If new session -- attach:

`database/schema.sql`

Claude needs table names to write correct vercel.json routes

`.env.example`

Claude needs variable names to reference in vercel.json headers

Copy-paste phrase:

```
NOVA Phase 0, Message 3.
# Sharing: database/schema.sql and .env.example
Please generate two files:
vercel.json
Routes: all /api/* to serverless functions, /admin/* as static files
Security headers on all routes:
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Referrer-Policy: strict-origin-when-cross-origin
package.json
name: nova-store
dependencies: better-sqlite3, jsonwebtoken, bcrypt,
@supabase/supabase-js, @upstash/redis, resend
scripts:
dev: run a local dev server on port 3000
create-admin: node database/create-admin.js
Generate both in one reply.
```

AFTER PHASE 0 -- WHAT YOU DO

1	**Run npm install** Open Command Prompt -> navigate to your nova-store folder: cd C:\Users\YourName\Desktop\nova-store npm install This installs all packages listed in package.json. Takes 1-2 minutes.
2	**Paste schema into Supabase** Go to supabase.com -> your project -> SQL Editor (left sidebar) Open database/schema.sql from your folder -> copy all the content Paste into the SQL editor -> click "Run" You should see "Success. No rows returned"
3	**Commit to GitHub** Push all new files to GitHub. Make sure .env and database/nova.db are NOT committed (they are in .gitignore)

PHASE 1

ADMIN LOGIN + ORDERS API + ORDERS PAGE

7 messages -- one page or feature per message

PART A -- THINGS YOU DO YOURSELF IN PHASE 1

1 **Create your admin account**

After Message 2 gives you database/create-admin.js, run: node database/create-admin.js your@email.com YourChosenPassword This saves a hashed password in the database. Do this ONCE.

2 **Test the login**

After Message 3, go to yoursite.vercel.app/admin/ Try logging in with the email and password you just created. If it works: you see the dashboard (even if it is empty for now).

3 **Remove EmailJS after Message 5**

After checkout.js is updated to use the API, you can remove the EmailJS script tags from index.html if you want. Or leave them -- the code no longer calls them so they do nothing.

4 **Test a full order**

Place a test order on your store. Then go to /admin/orders to see it.

PART B -- MESSAGES TO SEND CLAUDE

MSG 1-1 **api/_lib/ -- db.js + auth.js + rate-limit.js (the 3 shared utilities)**

Claude creates:

`api/_lib/db.js`

DB connector: SQLite locally, Supabase in production. Exports query()

`api/_lib/auth.js`

JWT verifier. Exports verifyAdmin(req) -- used by every admin route

`api/_lib/rate-limit.js`

Upstash Redis rate limiter. Exports rateLimit(ip,key,max,windowSec)

If new session -- attach:

`database/schema.sql`

Claude needs table names to write correct queries in db.js

Copy-paste phrase:

```
NOVA Phase 1, Message 1.
# Sharing: database/schema.sql
Build these 3 shared utility files in api/_lib/:
api/_lib/db.js
If NODE_ENV === "production": use @supabase/supabase-js client
```

```
with process.env.SUPABASE_URL and process.env.SUPABASE_SERVICE_KEY
If development: use better-sqlite3, file at database/nova.db
Export: async function query(sql, params=[]) that works for both
api/_lib/auth.js
Export: function verifyAdmin(req)
Read cookie named nova_admin_token from request headers
Verify with jwt.verify() using process.env.JWT_SECRET
Return the decoded payload, or throw an error
api/_lib/rate-limit.js
Export: async function rateLimit(ip, key, max, windowSec)
Use Upstash Redis REST API via fetch()
process.env.UPSTASH_REDIS_REST_URL and UPSTASH_REDIS_REST_TOKEN
Return: {allowed: boolean, remaining: number}
Generate one file per reply.
```

MSG 1-2 — api/auth/login.js + database/create-admin.js

Claude creates:

api/auth/login.js

POST /api/auth/login. bcrypt check, JWT cookie, 3-failure lockout.

database/create-admin.js

One-time Node.js script: creates hashed admin account in DB

If new session -- attach:

database/schema.sql

Needs admin_users table column names

api/_lib/db.js

Uses your query() function

api/_lib/rate-limit.js

Uses your rateLimit() function for lockout

Copy-paste phrase:

```
NOVA Phase 1, Message 2.
# Sharing: database/schema.sql, api/_lib/db.js, api/_lib/rate-limit.js
Build:
api/auth/login.js -- POST /api/auth/login
Read {email, password} from request body
Rate limit: 3 failures per IP -> block for 15 minutes
Query admin_users table for the email
Use bcrypt.compare() to check the password
On success: sign JWT with process.env.JWT_SECRET (expiry 8h)
set as HttpOnly cookie named nova_admin_token
return JSON {success: true}
On wrong password: return 401 with generic message
(never say if email or password is wrong -- just "Invalid credentials")
On lockout: return 429 with {error: "Too many attempts", retryAfter: N}
database/create-admin.js -- run from command line:
node database/create-admin.js admin@email.com YourPassword
Hash the password with bcrypt (10 rounds)
Insert into admin_users table
Print "Admin created: admin@email.com" on success
```

MSG 1-3 — admin/css/admin.css + admin/index.html (login page)

Claude creates:

`admin/css/admin.css`

Full admin CSS used by all 7 admin pages

`admin/index.html`

Login page: form, error display, lockout countdown, redirect on success

If new session -- attach:

`api/auth/login.js`

Claude needs to know the endpoint URL and response format

Copy-paste phrase:

```
NOVA Phase 1, Message 3.
# Sharing: api/auth/login.js
Build:
admin/css/admin.css -- dark theme CSS for all admin pages:
Background: #1A1A2E Sidebar: #0D0D1A Cards: #2D2D44
Gold accent #C9A96E for headings, active nav items, borders
Sidebar: 220px fixed left, links to 7 pages, logout at bottom
Tables: striped, hover highlight, sortable header arrows
Status badge colors:
pending=orange, accepted=blue, shipped=purple,
delivered=green, not_delivered=red, returned=gray
Buttons: primary (gold bg), danger (red bg), secondary (gray)
Responsive: sidebar becomes top nav on mobile
admin/index.html -- LOGIN PAGE (no sidebar):
Centered card on dark background
NOVA logo text (gold, large)
Email input + password input with eye icon to toggle visibility
Login button
Error div (hidden by default, shown on failed login)
On success (200 from POST /api/auth/login): redirect to dashboard.html
On 3 failures: show "Too many attempts. Try again in X minutes." with countdown
Inline JavaScript only -- no separate .js file for this page
```

MSG 1-4 api/orders/index.js + api/orders/[id].js

Claude creates:

`api/orders/index.js`

GET: admin orders list with filters. POST: new order from store, replaces EmailJS.

`api/orders/[id].js`

PATCH: update order status (admin). DELETE: remove order (admin).

If new session -- attach:

`database/schema.sql`

Needs orders table column names exactly

`api/_lib/db.js`

Uses your query() function

`api/_lib/auth.js`

Uses verifyAdmin() on GET, PATCH, DELETE

`api/_lib/rate-limit.js`

Uses rateLimit() on POST to stop spam orders

Copy-paste phrase:

```
NOVA Phase 1, Message 4.
# Sharing: database/schema.sql, api/_lib/db.js,
# api/_lib/auth.js, api/_lib/rate-limit.js
Build:
api/orders/index.js:
```

```
GET /api/orders -- requires verifyAdmin()
Query params: status, page (default 1), limit (default 20),
search (match customer_name OR customer_phone OR order_number),
date_from, date_to
Returns: {orders:[...], total, page, totalPages}
POST /api/orders -- PUBLIC (called by store checkout)
Rate limit: 5 per IP per hour
Body: {name, address, phone, cart:[{id,name,qty,size,color}], lang}
Validate: name/address/phone not empty, max 200 chars each
Validate: cart is array, not empty, each item has id and qty
Fetch product prices from products table -- calculate total server-side
Never trust price from client
Generate order_number: "NOVA-" + 6 random digits
Save to orders table with status="pending"
Send email via Resend to process.env.RESEND_TO_EMAIL with order details
Return: {success:true, orderId: order_number}
api/orders/[id].js:
PATCH -- requires verifyAdmin()
Body: {status} -- validate against allowed values:
pending, accepted, shipped, delivered, not_delivered, returned
DELETE -- requires verifyAdmin()
```

MSG 1-5 **checkout.js (updated -- replaces EmailJS)**

Claude creates:

checkout.js

Updated: placeOrder() now POSTs to /api/orders instead of emailjs.send()

If new session -- attach:

checkout.js

The file being changed -- Claude edits it directly

api/orders/index.js

Claude needs the exact POST body format and response format

Copy-paste phrase:

```
NOVA Phase 1, Message 5.
# Sharing: checkout.js and api/orders/index.js
Update ONLY the placeOrder() function in checkout.js.
Replace the emailjs.send() call with a fetch to /api/orders:
const res = await fetch("/api/orders", {
method: "POST",
headers: {"Content-Type": "application/json"},
body: JSON.stringify({
name: customerName,
address: customerAddress,
phone: customerPhone,
cart: cart.map(item => ({
id: item.id,
name: typeof item.name === "object" ? item.name[currentLang] : item.name,
qty: item.qty,
size: item.size,
color: item.color
})),
lang: currentLang
})
})
const data = await res.json()
```

```
if (!data.success) throw new Error(data.error || "Order failed")
Keep the same success/error behavior that already exists.
Do not change the form, the validation, the cart logic, or the UI.
Return the full updated checkout.js.
```

MSG 1-6 admin/dashboard.html + admin/js/admin-auth.js

Claude creates:

admin/dashboard.html

Overview: stat cards, status counts, recent 10 orders, low stock alerts

admin/js/admin-auth.js

Shared across all admin pages: JWT check on load, getAuthHeaders(), logout()

If new session -- attach:

admin/css/admin.css

Dashboard uses this CSS -- Claude must know the class names

api/_lib/auth.js

Claude uses the same JWT approach in admin-auth.js

api/orders/index.js

Dashboard fetches from GET /api/orders -- needs response format

Copy-paste phrase:

```
NOVA Phase 1, Message 6.
# Sharing: admin/css/admin.css, api/_lib/auth.js, api/orders/index.js
Build:
admin/js/admin-auth.js (used by ALL admin pages):
On DOMContentLoaded: check for cookie named nova_admin_token
If missing: redirect to /admin/index.html
If present but JWT expired: redirect to /admin/index.html
Export getAuthHeaders() -> returns {Authorization: "Bearer " + token}
Export logout() -> clear cookie, redirect to /admin/index.html
admin/dashboard.html:
Include admin/css/admin.css and admin/js/admin-auth.js
Sidebar: nav links to all 7 pages + logout button at bottom
4 stat cards: Orders Today, Total Revenue, Pending Orders, Active Products
Status row: 6 badges showing count for each status (fetched from API)
Recent 10 orders mini-table: order number, customer name, total, status, date
Low stock section: list products where stock <= 5 (from GET /api/products)
Everything fetched on page load with getAuthHeaders()
```

MSG 1-7 admin/orders.html + admin/js/admin-orders.js

Claude creates:

admin/orders.html

Orders management: status tabs, search, date filter, row expand, bulk, CSV

admin/js/admin-orders.js

All orders page logic: fetch, filter, PATCH status, DELETE, CSV export

If new session -- attach:

admin/css/admin.css

Must use the same CSS classes for consistent styling

admin/js/admin-auth.js

All admin pages use this for auth

api/orders/index.js

Needs GET query params: status, search, date_from, date_to, page

api/orders/[id].js

Needs PATCH body format and DELETE endpoint

Copy-paste phrase:

```
NOVA Phase 1, Message 7.
# Sharing: admin/css/admin.css, admin/js/admin-auth.js,
# api/orders/index.js, api/orders/[id].js
Build admin/orders.html and admin/js/admin-orders.js:
Tabs at top: All | Pending | Accepted | Shipped | Delivered |
Not Delivered | Returned -- each shows count in ()
Search bar: searches customer name, phone, order number
Date range: from-date and to-date inputs
Table columns: checkbox, Order#, Customer Name, Phone,
Items, Total, Date, Status badge, Actions
Click any row: expand below the row to show full details:
Customer full address
Each item: product name, size, color, quantity, unit price
Per-row actions:
Status dropdown (all 6 options) + Save -> PATCH /api/orders/[id]
Delete button -> confirm dialog -> DELETE /api/orders/[id]
Row updates in place without page reload
Bulk actions bar (shows when any checkbox is checked):
"X orders selected" + status dropdown + Apply button
Loops through selected IDs and patches each
Revenue total: sum of filtered orders shown below the table
Export CSV button: downloads all filtered orders as .csv
Columns: order_number, name, phone, address, items_summary, total, status, date
Pagination: prev/next + page X of Y
```

AFTER PHASE 1 -- WHAT YOU DO

1	**Apply schema in Supabase** If not done yet, paste database/schema.sql into Supabase SQL editor and run it.
2	**Commit and deploy** Push all new files to GitHub -> Vercel auto-deploys in ~20 seconds.
3	**Create your admin account** In Command Prompt in your project folder: node database/create-admin.js your@email.com YourPassword
4	**Test the login** Go to yoursite.vercel.app/admin/ and log in.
5	**Place a test order** Use your store to place a fake order. Check /admin/orders to see it appear.

PHASE 2

ADMIN PRODUCTS PAGE + PRODUCTS API

3 messages

PART A -- THINGS YOU DO IN PHASE 2

1 **Run seed.sql in Supabase**

After Message 2 gives you database/seed.sql: Go to Supabase -> SQL Editor -> paste the full content of seed.sql -> click Run This imports all your products into the database. Do this ONCE.

2 **Test products in admin**

After Message 3, go to /admin/products to see your products listed. Try editing a product price and check the store to see it updated.

PART B -- MESSAGES

MSG 2-1 **api/products/index.js + api/products/[id].js**

Claude creates:

`api/products/index.js`

GET public full product list with colors+images+sizes. POST admin add.

`api/products/[id].js`

PATCH admin edit any fields. DELETE soft or hard.

If new session -- attach:

`database/schema.sql`

Needs products, product_colors, product_images, product_sizes columns

`api/_lib/db.js`

Uses query()

`api/_lib/auth.js`

Uses verifyAdmin() for POST, PATCH, DELETE

Copy-paste phrase:

```
NOVA Phase 2, Message 1.
# Sharing: database/schema.sql, api/_lib/db.js, api/_lib/auth.js
Build:
api/products/index.js:
GET /api/products -- PUBLIC
Return all products where active=true with nested data:
colors array (each color has name_en, name_ar, hex, images array)
sizes array (each size has size_label and stock)
Query params: category (slug), search (name_en or name_ar contains)
sort: price-asc, price-desc, discount (has old_price)
POST /api/products -- requires verifyAdmin()
Body: {name_en, name_ar, desc_en, desc_ar, category_id,
price, old_price, stock, active,
colors:[{name_en,name_ar,hex,images:[url,...]},...],
sizes:[{size_label,stock},...]}
Insert into products, then product_colors, product_images, product_sizes
Return: {success:true, productId}
```

```
api/products/[id].js:
PATCH -- requires verifyAdmin()
Accept any subset of product fields
If colors array provided: delete existing colors/images, insert new ones
If sizes array provided: delete existing sizes, insert new ones
DELETE -- requires verifyAdmin()
?permanent=true: delete product and all related rows
default (no param): set active=false (hidden from store, data kept)
```

MSG 2-2 database/seed.sql -- import your products

Claude creates:

database/seed.sql

Complete INSERT SQL for all products from your products.js -- run once in Supabase

If new session -- attach:

products.js

Your actual product data -- Claude reads this to write the SQL

database/schema.sql

Claude needs exact column names to write correct INSERT statements

Copy-paste phrase:

```
NOVA Phase 2, Message 2.
# Sharing: products.js and database/schema.sql
Generate database/seed.sql:
First: INSERT OR IGNORE the categories:
Look at my products.js category values and create matching rows
Then for each product in products.js:
INSERT OR IGNORE into products (name_en, name_ar, desc_en, desc_ar,
price, old_price, stock, active, category_id)
INSERT OR IGNORE into product_colors for each color
INSERT OR IGNORE into product_images for each image URL
INSERT OR IGNORE into product_sizes for each size with its stock
Use INSERT OR IGNORE so running twice creates no duplicates
Add a comment above each product block showing the product name
I will paste this entire file into Supabase SQL editor and click Run once
```

MSG 2-3 admin/products.html + admin/js/admin-products.js

Claude creates:

admin/products.html

Products list with search/filter. Slide-in add/edit panel. Delete.

admin/js/admin-products.js

Full CRUD logic: fetch, render, add, edit, delete, color/size editor

If new session -- attach:

admin/css/admin.css

Must use same CSS classes

admin/js/admin-auth.js

All admin pages use this

api/products/index.js

Needs GET and POST endpoint format

api/products/[id].js

Needs PATCH and DELETE format

database/schema.sql

Helps understand product structure

Copy-paste phrase:

```
NOVA Phase 2, Message 3.
# Sharing: admin/css/admin.css, admin/js/admin-auth.js,
# api/products/index.js, api/products/[id].js, database/schema.sql
Build admin/products.html and admin/js/admin-products.js:
Product list (table):
Columns: thumbnail (first image), name EN, category, price,
old_price if set, stock badge, active toggle, Edit, Delete
Stock badge: red if 0, orange if <=5, green otherwise
Active toggle: switch that calls PATCH with {active: bool}
Search input + category dropdown filter at top
Add Product button (top right) -> opens slide panel
Slide-in panel (right side, overlays page):
Title: "Add Product" or "Edit Product"
Name EN + Name AR (inputs)
Description EN + Description AR (textareas)
Category (dropdown -- loads from GET /api/categories)
Price + Old Price (number inputs -- Old Price optional)
Total Stock (number input)
Colors section:
"Add Color" button creates a new color row containing:
Name EN input, Name AR input, hex color picker input,
textarea for image URLs (one per line)
"Remove" button per color row
Sizes section (updates when category changes):
Checkbox list of that category default sizes
When a size is checked: show a stock number input next to it
Active checkbox
Save (POST or PATCH) + Cancel buttons
Edit: click Edit button -> fill panel with existing data -> PATCH on save
Delete: confirm dialog -> DELETE /api/products/[id]?permanent=true
```

AFTER PHASE 2

1	**Run seed.sql** Supabase -> SQL Editor -> paste seed.sql content -> Run
2	**Test products admin** Go to /admin/products -- you should see all your products
3	**Test adding a product** Add a test product -> check the store to see it appear

PHASE 3

ADMIN CATEGORIES + STORE LOADS FROM API

3 messages

PART A -- THINGS YOU DO IN PHASE 3

1 **Test after Message 3**

After index.html is updated, open your store and confirm products still load correctly. They should now come from the database instead of products.js.

2 **Add ALLOWED_ORIGIN to Vercel**

In Vercel -> Settings -> Environment Variables Add: ALLOWED_ORIGIN = https://yourstore.vercel.app (your actual Vercel URL) This is used by the CORS settings built in Phase 5.

PART B -- MESSAGES

MSG 3-1 **api/categories/index.js**

Claude creates:

`api/categories/index.js`

GET public list with sizes. POST/PATCH/DELETE admin.

If new session -- attach:

`database/schema.sql`

Needs categories and category_sizes table structure

`api/_lib/db.js`

Uses query()

`api/_lib/auth.js`

Uses verifyAdmin() for POST, PATCH, DELETE

Copy-paste phrase:

```
NOVA Phase 3, Message 1.
# Sharing: database/schema.sql, api/_lib/db.js, api/_lib/auth.js
Build api/categories/index.js:
GET /api/categories -- PUBLIC
Return all categories with their default_sizes array
Example row: {id:1, name_en:"Shoes", name_ar:"...",
slug:"shoes", default_sizes:["36","37","38","39","40"]}
POST /api/categories -- requires verifyAdmin()
Body: {name_en, name_ar, slug, default_sizes:[...]}
Insert into categories, then insert each size into category_sizes
PATCH /api/categories?id=X -- requires verifyAdmin()
Body: any subset of {name_en, name_ar, slug, default_sizes}
If default_sizes provided: delete existing, insert new
DELETE /api/categories?id=X -- requires verifyAdmin()
First check: count products with this category_id
If count > 0: return 400 error "X products use this category"
If 0: delete category_sizes rows, then delete category
```

MSG 3-2 admin/categories.html + admin/js/admin-categories.js

Claude creates:

`admin/categories.html`

Categories list with inline edit + sizes-per-category editor

`admin/js/admin-categories.js`

Full categories and sizes management logic

If new session -- attach:

`admin/css/admin.css`

Same CSS

`admin/js/admin-auth.js`

Same auth

`api/categories/index.js`

Needs all 4 endpoint formats

Copy-paste phrase:

```
NOVA Phase 3, Message 2.
# Sharing: admin/css/admin.css, admin/js/admin-auth.js,
# api/categories/index.js
Build admin/categories.html and admin/js/admin-categories.js:
Section 1 -- Categories:
Table: Name EN, Name AR, Slug, Product count, Edit, Delete
Edit: click -> cells become input fields -> Save / Cancel inline
Delete: disabled with tooltip if product count > 0
Add row at bottom: 3 inputs + Add button
Section 2 -- Default Sizes Per Category:
One row per category showing current sizes as chips [36] [37] [38]
Edit button: opens inline chip editor below that row
Type a size -> press Enter or comma -> adds chip
Click X on chip -> removes it
Save: PATCH /api/categories?id=X with new default_sizes array
Cancel: closes editor, reverts
```

MSG 3-3 index.html (update init() to load from API)

Claude creates:

`index.html`

Updated: init() fetches /api/products on load. Static array becomes fallback.

If new session -- attach:

`index.html`

The file being changed -- Claude edits init() only

`api/products/index.js`

Claude needs the GET response format to map fields correctly

Copy-paste phrase:

```
NOVA Phase 3, Message 3.
# Sharing: index.html and api/products/index.js
In index.html, find the init() function in the inline script at the bottom.
Keep the existing static products array but rename it to PRODUCTS_STATIC.
Set: let products = PRODUCTS_STATIC; (fallback -- works if API fails)
Add an async loadProducts() function:
async function loadProducts() {
try {
const res = await fetch("/api/products")
if (!res.ok) throw new Error("API error")
const data = await res.json()
```

```
products = data
} catch(e) {
products = PRODUCTS_STATIC // silent fallback
}
renderProducts()
setupFilters()
}
In init(): replace the renderProducts() call with loadProducts()
Show a loading state (spinner or skeleton) before loadProducts() resolves
Change ONLY init() and add loadProducts() -- touch nothing else.
Return the complete updated index.html.
```

PHASE 4

ADMIN REVIEWS + NEWSLETTER

6 messages

PART A -- THINGS YOU DO IN PHASE 4

1 **Test newsletter signup**

After Message 2 and 4, sign up for the newsletter on the store. Go to /admin/newsletter to see the subscriber appear.

2 **Test a review**

After Message 1 and 3, submit a review on the store. Go to /admin/reviews -> Pending tab -> approve it. Refresh the store -- the review should appear.

PART B -- MESSAGES

MSG 4-1 **api/reviews/index.js**

Claude creates:

`api/reviews/index.js`

GET approved (public). POST submit. PATCH approve/edit. DELETE.

If new session -- attach:

`database/schema.sql`

Needs reviews table columns

`api/_lib/db.js`

Uses query()

`api/_lib/auth.js`

verifyAdmin() for PATCH and DELETE

`api/_lib/rate-limit.js`

Rate limit on POST

Copy-paste phrase:

```
NOVA Phase 4, Message 1.
# Sharing: database/schema.sql, api/_lib/db.js,
# api/_lib/auth.js, api/_lib/rate-limit.js
Build api/reviews/index.js:
GET /api/reviews -- PUBLIC
If no admin JWT: return only verified=true rows
If valid admin JWT: return ALL rows (for moderation)
Optional query param: product_id
Order by created_at DESC
POST /api/reviews -- PUBLIC
Rate limit: 1 review per product_id per IP per 24 hours
Body: {product_id, stars, text_en, text_ar, author}
Validate: stars 1-5, text_en not empty, author not empty
Save with verified=false
Return: {success:true}
```

```
PATCH /api/reviews?id=X -- requires verifyAdmin()
Body: {verified?, text_en?, text_ar?}
Update only provided fields
DELETE /api/reviews?id=X -- requires verifyAdmin()
```

MSG 4-2 api/newsletter/index.js

Claude creates:

`api/newsletter/index.js`

POST subscribe (public). GET list (admin). DELETE (admin).

If new session -- attach:

`database/schema.sql`

Needs newsletter_subscribers table

`api/_lib/db.js`

Uses query()

`api/_lib/auth.js`

verifyAdmin() for GET and DELETE

`api/_lib/rate-limit.js`

Rate limit on POST

Copy-paste phrase:

```
NOVA Phase 4, Message 2.
# Sharing: database/schema.sql, api/_lib/db.js,
# api/_lib/auth.js, api/_lib/rate-limit.js
Build api/newsletter/index.js:
POST /api/newsletter -- PUBLIC
Rate limit: 3 per IP per hour
Body: {email, lang}
Validate: valid email format, lang must be "en" or "ar"
If email already exists: return {success:true} silently
If new: insert into newsletter_subscribers
Return: {success:true}
GET /api/newsletter -- requires verifyAdmin()
Query params: search (email contains), lang, page, limit
Returns: {subscribers:[...], total}
DELETE /api/newsletter?id=X -- requires verifyAdmin()
```

MSG 4-3 reviews.js (updated)

Claude creates:

`reviews.js`

Updated: loads reviews from /api/reviews. Falls back to REVIEWS_DATA if API fails.

If new session -- attach:

`reviews.js`

The file being changed

`api/reviews/index.js`

Claude needs the GET response format to map fields

Copy-paste phrase:

```
NOVA Phase 4, Message 3.
# Sharing: reviews.js and api/reviews/index.js
Update reviews.js -- change only the renderReviews() function:
At the start of renderReviews(), fetch from API:
try {
```

```
const res = await fetch("/api/reviews")
const data = await res.json()
reviewsData = data // use fetched array
} catch(e) {
reviewsData = REVIEWS_DATA // fallback to existing array
}
// continue with existing rendering logic using reviewsData
Keep the REVIEWS_DATA array in the file as fallback.
Keep renderStars() and all other functions unchanged.
Return the complete updated reviews.js file.
```

MSG 4-4 engagement.js (updated)

Claude creates:

`engagement.js`

Updated: newsletter form POSTs to /api/newsletter instead of placeholder

If new session -- attach:

`engagement.js`

The file being changed

`api/newsletter/index.js`

Needs the POST body format and success response

Copy-paste phrase:

```
NOVA Phase 4, Message 4.
# Sharing: engagement.js and api/newsletter/index.js
In engagement.js, find the newsletter form submit handler.
It is the event listener that handles the email input form submission.
Replace the current submission logic with:
fetch("/api/newsletter", {
method: "POST",
headers: {"Content-Type":"application/json"},
body: JSON.stringify({email: emailValue, lang: currentLang})
})
.then(r => r.json())
.then(data => {
if (data.success) { /* show same success toast as before */ }
else throw new Error()
})
.catch(() => { /* show same error toast as before */ })
Keep all toast/UI behavior identical to what already exists.
Do not change any other part of engagement.js.
Return the complete updated engagement.js.
```

MSG 4-5 admin/reviews.html + admin/js/admin-reviews.js

Claude creates:

`admin/reviews.html`

Pending and approved tabs. Per-review: approve, edit text, reject, hide, delete.

`admin/js/admin-reviews.js`

All reviews moderation logic

If new session -- attach:

`admin/css/admin.css`

Same CSS

`admin/js/admin-auth.js`

Same auth

`api/reviews/index.js`

Needs GET (all for admin), PATCH, DELETE endpoints

Copy-paste phrase:

```
NOVA Phase 4, Message 5.
# Sharing: admin/css/admin.css, admin/js/admin-auth.js,
# api/reviews/index.js
Build admin/reviews.html and admin/js/admin-reviews.js:
Fetch all reviews with admin JWT (returns both verified and unverified)
Split into two tabs: "Pending (N)" and "Approved (N)"
Each review card shows:
Star rating (visual gold stars)
Reviewer name + date
Product name
Review text EN and AR (both shown)
Pending tab -- actions per card:
Approve: PATCH {verified:true}
Edit: text_en and text_ar become editable inputs
Save+Approve: PATCH {verified:true, text_en:..., text_ar:...}
Reject: confirm dialog -> DELETE
Approved tab -- actions per card:
Hide: PATCH {verified:false} -> moves back to pending
Delete: confirm -> DELETE
```

MSG 4-6 admin/newsletter.html + admin/js/admin-newsletter.js

Claude creates:

`admin/newsletter.html`

Subscriber list: search, lang filter, CSV export, delete

`admin/js/admin-newsletter.js`

Newsletter page logic

If new session -- attach:

`admin/css/admin.css`

Same CSS

`admin/js/admin-auth.js`

Same auth

`api/newsletter/index.js`

Needs GET endpoint params and DELETE endpoint

Copy-paste phrase:

```
NOVA Phase 4, Message 6.
# Sharing: admin/css/admin.css, admin/js/admin-auth.js,
# api/newsletter/index.js
Build admin/newsletter.html and admin/js/admin-newsletter.js:
Stats row: Total subscribers | EN: X | AR: X
Search input: filter by email (live filter)
Language tabs: All / EN / AR
Table columns: email, language, signup date, Delete button
Delete: no confirm needed -- DELETE /api/newsletter?id=X
Remove row instantly from table without reload
Export CSV button: download filtered results as .csv
Columns: email, language, signup_date
Pagination if total > 50
```

PHASE 5

SECURITY HARDENING + FINAL AUDIT

3 messages

PART A -- THINGS YOU DO IN PHASE 5

1 **Add ALLOWED_ORIGIN to Vercel**

If not done in Phase 3: Vercel -> your project -> Settings -> Environment Variables Add: ALLOWED_ORIGIN = https://yourstore.vercel.app

2 **Run RLS SQL in Supabase**

After Message 2, paste the RLS SQL into Supabase SQL Editor and run it. Then go to Supabase -> Table Editor -> click any table -> check "RLS Enabled" badge.

3 **Final test**

Place a real test order -> check admin orders -> check email received Add a product in admin -> check store -> Edit a product -> confirm store updates

PART B -- MESSAGES

MSG 5-1 **Security headers + CORS + api/_lib/cors.js**

Claude creates:	If new session -- attach:
`vercel.json`	`vercel.json`
Updated with security headers for all routes	Being updated
`api/_lib/cors.js`	`api/orders/index.js`
CORS middleware: locks API to your domain only	Example route -- Claude shows where to add CORS helper

Copy-paste phrase:

```
NOVA Phase 5, Message 1.
# Sharing: vercel.json, api/orders/index.js
Add security:
In vercel.json, add headers block for all routes:
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Referrer-Policy: strict-origin-when-cross-origin
Permissions-Policy: camera=(), microphone=()
Create api/_lib/cors.js:
Export setCORS(res, req):
Allowed origin: process.env.ALLOWED_ORIGIN
Set Access-Control-Allow-Origin
Set Access-Control-Allow-Methods: GET,POST,PATCH,DELETE,OPTIONS
Set Access-Control-Allow-Headers: Content-Type
Handle OPTIONS preflight: return 200 immediately
```

```
Update api/orders/index.js to call setCORS at the top -- show the pattern
I will apply the same pattern to other routes manually
```

MSG 5-2 Supabase Row Level Security SQL

Claude creates:

`supabase_rls.sql`

RLS policies for all tables -- paste and run in Supabase SQL editor

If new session -- attach:

`database/schema.sql`

Needs all table names and column names

Copy-paste phrase:

```
NOVA Phase 5, Message 2.
# Sharing: database/schema.sql
Generate supabase_rls.sql with Row Level Security for all tables:
First: ALTER TABLE X ENABLE ROW LEVEL SECURITY; for all tables
Then policies:
orders:
SELECT -- service role only (API reads orders, customers cannot)
INSERT -- allow all (customers place orders through API)
products:
SELECT -- allow all where active=true (public store)
INSERT/UPDATE/DELETE -- service role only
categories:
SELECT -- allow all (public navigation)
INSERT/UPDATE/DELETE -- service role only
reviews:
SELECT -- allow all where verified=true (service role sees all)
INSERT -- allow all (customers submit reviews)
UPDATE/DELETE -- service role only
newsletter_subscribers:
INSERT -- allow all (customers subscribe)
SELECT/DELETE -- service role only
admin_users:
ALL -- service role only -- never accessible publicly
product_colors, product_images, product_sizes, category_sizes:
SELECT -- allow all (needed for public product loading)
INSERT/UPDATE/DELETE -- service role only
Add comments explaining each policy
```

MSG 5-3 Input validation audit + security checklist

Claude creates:	If new session -- attach:
`All api/ route files`	`api/orders/index.js`
Updated with any missing validation	Review order inputs
`SECURITY_CHECKLIST.md`	`api/auth/login.js`
Final manual verification checklist	Review login inputs
	`api/products/index.js`
	Review product inputs
	`api/reviews/index.js`
	Review review inputs
	`api/newsletter/index.js`
	Review newsletter inputs

Copy-paste phrase:

```
NOVA Phase 5, Message 3.
# Sharing: all 5 api/ route files listed above
Do a full input validation audit on every route file:
For each input field check:
- String fields: not empty, max length (200 chars default)
- Number fields: is number, not negative, sensible range
- Email: validate format with regex
- Enum fields (status, lang): check against explicit allowed list
- Array fields: is array, not empty, check each item
Fix any missing validation -- return updated file content
Also generate SECURITY_CHECKLIST.md -- a final checklist:
- All 8 Vercel env vars set
- ALLOWED_ORIGIN set to real domain
- RLS enabled on all Supabase tables
- Admin account created with strong password
- Test order placed and email received
- HTTPS confirmed on Vercel
- .env in .gitignore
- database/nova.db in .gitignore
- analytics.js debug:false
- Admin login tested: wrong password lockout works
```

CONTEXT PHRASE -- PASTE AT START OF EVERY NEW SESSION

Paste this FIRST, then add the per-message phrase, then attach the listed files.

```
I am building NOVA -- a bilingual (EN+AR) fashion e-commerce store.
Frontend: 18 static HTML/CSS/JS files deployed on Vercel.
Backend stack:
Vercel Serverless Functions (Node.js)
SQLite for local development (file: database/nova.db)
Supabase PostgreSQL for production
Resend for order notification emails
Upstash Redis for rate limiting
Admin dashboard at /admin/ -- 7 pages:
login (index.html), dashboard, orders, products,
categories, reviews, newsletter
Shared: admin/css/admin.css + admin/js/admin-auth.js
API routes under /api/ as Vercel serverless functions.
Shared utilities: api/_lib/db.js, auth.js, rate-limit.js
I am on Phase X, Message Y.
[Paste the specific message phrase below this line]
```

ALL MESSAGES SUMMARY

Msg	Builds	Share if new session
P0-1	database/schema.sql	Nothing
P0-2	.env.example + .gitignore	schema.sql
P0-3	vercel.json + package.json	schema.sql + .env.example
P1-1	api/_lib/db.js + auth.js + rate-limit.js	schema.sql
P1-2	api/auth/login.js + create-admin.js	schema.sql + db.js + rate-limit.js
P1-3	admin/css/admin.css + admin/index.html	api/auth/login.js
P1-4	api/orders/index.js + [id].js	schema.sql + db.js + auth.js + rate-limit.js
P1-5	checkout.js (updated)	checkout.js + api/orders/index.js
P1-6	admin/dashboard.html + admin-auth.js	admin.css + api/_lib/auth.js + api/orders/index.js
P1-7	admin/orders.html + admin-orders.js	admin.css + admin-auth.js + api/orders/index.js + api/orders/[id].js
P2-1	api/products/index.js + [id].js	schema.sql + db.js + auth.js
P2-2	database/seed.sql	products.js + schema.sql
P2-3	admin/products.html + admin-products.js	admin.css + admin-auth.js + api/products/index.js + [id].js + schema.sql
P3-1	api/categories/index.js	schema.sql + db.js + auth.js

P3-2	admin/categories.html + admin-categories.js	admin.css + admin-auth.js + api/categories/index.js
P3-3	index.html (updated)	index.html + api/products/index.js
P4-1	api/reviews/index.js	schema.sql + db.js + auth.js + rate-limit.js
P4-2	api/newsletter/index.js	schema.sql + db.js + auth.js + rate-limit.js
P4-3	reviews.js (updated)	reviews.js + api/reviews/index.js
P4-4	engagement.js (updated)	engagement.js + api/newsletter/index.js
P4-5	admin/reviews.html + admin-reviews.js	admin.css + admin-auth.js + api/reviews/index.js
P4-6	admin/newsletter.html + admin-newsletter.js	admin.css + admin-auth.js + api/newsletter/index.js
P5-1	vercel.json (updated) + api/_lib/cors.js	vercel.json + api/orders/index.js
P5-2	supabase_rls.sql	schema.sql
P5-3	Validation fixes + SECURITY_CHECKLIST.md	All 5 api/ route files

NOVA . 26 messages . 5 phases . Full backend

Context phrase + correct files for every message = Claude always has everything it needs