NOVA

Frontend Changes Guide

Full project structure | 4 frontend messages | Exact files to attach

Companion to NOVA_Guide.pdf -- use both together

How to use this PDF This is the companion to NOVA_Guide.pdf. That guide builds the entire backend (26 messages). This guide has the 4 frontend messages that connect your store to it. Use the same context phrase from NOVA_Guide.pdf before each message. The full file structure on the next pages shows every file, what folder it lives in, and when it is created.

YES -- THE FRONTEND IS BUILT FOR THIS BACKEND. HERE IS HOW THEY CONNECT.

Frontend action	Calls this backend endpoint	Built in
Customer places order	POST /api/orders	NOVA Guide P1-4 + Frontend MSG 1-5
Store homepage loads products	GET /api/products	NOVA Guide P2-1 + Frontend MSG 3-3
Reviews section loads	GET /api/reviews	NOVA Guide P4-1 + Frontend MSG 4-3
Newsletter form submits	POST /api/newsletter	NOVA Guide P4-2 + Frontend MSG 4-4
Admin login	POST /api/auth/login	NOVA Guide P1-2 + P1-3 (admin only)
Cart saved between visits	localStorage -- no backend needed	Already works
Wishlist saved	localStorage -- no backend needed	Already works
Language / theme	localStorage -- no backend needed	Already works

[!] The 4 frontend messages in this PDF match the phase numbers in NOVA_Guide.pdf. MSG 1-5 here is the same as P1-5 in the main guide. MSG 3-3 = P3-3. MSG 4-3 = P4-3. MSG 4-4 = P4-4.

FULL STRUCTURE

COMPLETE FILE STRUCTURE -- EVERY FILE IN EVERY FOLDER

Green = NEW file built by Claude | Orange = EXISTING file that gets a small change | Gray = untouched

File	What it does	Status	Created in
nova-store/	ROOT FOLDER -- your project		
`index.html`	Your store homepage -- product grid, cart, modal, checkout	**CHANGES**	Phase 3, MSG 3-3
`styles.css`	All store CSS	UNTOUCHED	--
`extras.css`	Extra store CSS	UNTOUCHED	--
`products.js`	Static products array -- KEPT as fallback after API	UNTOUCHED	--
`shop.js`	Product grid rendering -- reads global products variable	UNTOUCHED	--
`modal.js`	Product detail modal	UNTOUCHED	--
`cart.js`	Cart + wishlist logic -- uses localStorage	UNTOUCHED	--
`checkout.js`	Checkout form -- placeOrder() sends to /api/orders	**CHANGES**	Phase 1, MSG 1-5
`reviews.js`	Reviews section -- fetches from /api/reviews	**CHANGES**	Phase 4, MSG 4-3
`engagement.js`	Popups, newsletter -- form sends to /api/newsletter	**CHANGES**	Phase 4, MSG 4-4
`analytics.js`	GA4 + Plausible tracking	UNTOUCHED	--
`lang.js`	Language switching EN/AR	UNTOUCHED	--
`i18n.js`	Translation strings	UNTOUCHED	--
`darktheme.js`	Dark/light mode toggle	UNTOUCHED	--
`trustbar.js`	Trust badge strip	UNTOUCHED	--
`infosections.js`	Delivery info + returns sections	UNTOUCHED	--
admin/	ADMIN FOLDER -- all admin pages		
`admin/index.html`	Admin login page	**NEW**	Phase 1, MSG 1-3
`admin/dashboard.html`	Stats + recent orders + low stock	**NEW**	Phase 1, MSG 1-6
`admin/orders.html`	Full orders management	**NEW**	Phase 1, MSG 1-7
`admin/products.html`	Products CRUD with color/size editor	**NEW**	Phase 2, MSG 2-3
`admin/categories.html`	Categories + default sizes	**NEW**	Phase 3, MSG 3-2
`admin/reviews.html`	Reviews moderation -- approve/reject	**NEW**	Phase 4, MSG 4-5
`admin/newsletter.html`	Subscriber list + CSV export	**NEW**	Phase 4, MSG 4-6
admin/css/	Admin CSS folder		

`admin/css/admin.css`	All admin CSS -- dark theme, sidebar, tables, badges	NEW	Phase 1, MSG 1-3
admin/js/	Admin JS folder		
`admin/js/admin-auth.js`	JWT check on load, getAuthHeaders(), logout() -- used by all 7 pages	NEW	Phase 1, MSG 1-6
`admin/js/admin-orders.js`	Orders page logic	NEW	Phase 1, MSG 1-7
`admin/js/admin-products.js`	Products page logic	NEW	Phase 2, MSG 2-3
`admin/js/admin-categories.js`	Categories page logic	NEW	Phase 3, MSG 3-2
`admin/js/admin-reviews.js`	Reviews page logic	NEW	Phase 4, MSG 4-5
`admin/js/admin-newsletter.js`	Newsletter page logic	NEW	Phase 4, MSG 4-6
api/	API FOLDER -- Vercel serverless functions		
api/_lib/	Shared utilities used by all route files		
`api/_lib/db.js`	DB connector: SQLite (dev) / Supabase (prod)	NEW	Phase 1, MSG 1-1
`api/_lib/auth.js`	JWT verifier -- verifyAdmin(req)	NEW	Phase 1, MSG 1-1
`api/_lib/rate-limit.js`	Upstash Redis rate limiter	NEW	Phase 1, MSG 1-1
`api/_lib/cors.js`	CORS: locks API to your domain	NEW	Phase 5, MSG 5-1
api/auth/	Auth routes		
`api/auth/login.js`	POST -- admin login, bcrypt, JWT cookie	NEW	Phase 1, MSG 1-2
api/orders/	Orders routes		
`api/orders/index.js`	GET admin list | POST new order (store sends here)	NEW	Phase 1, MSG 1-4
`api/orders/[id].js`	PATCH status | DELETE order	NEW	Phase 1, MSG 1-4
api/products/	Products routes		
`api/products/index.js`	GET public list | POST admin add (store reads this)	NEW	Phase 2, MSG 2-1
`api/products/[id].js`	PATCH admin edit | DELETE	NEW	Phase 2, MSG 2-1
api/categories/	Categories routes		
`api/categories/index.js`	GET public | POST/PATCH/DELETE admin	NEW	Phase 3, MSG 3-1
api/reviews/	Reviews routes		
`api/reviews/index.js`	GET approved | POST submit | PATCH approve | DELETE	NEW	Phase 4, MSG 4-1
api/newsletter/	Newsletter routes		
`api/newsletter/index.js`	POST subscribe | GET list | DELETE	NEW	Phase 4, MSG 4-2

database/	Database files		
`database/schema.sql`	All 11 CREATE TABLE statements	**NEW**	Phase 0, MSG 0-1
`database/seed.sql`	INSERT all products from products.js -- run once	**NEW**	Phase 2, MSG 2-2
`database/create-admin.js`	One-time script: node create-admin.js email pass	**NEW**	Phase 1, MSG 1-2
`database/nova.db`	SQLite file -- LOCAL ONLY -- never goes to GitHub	**NEW**	Auto-created
`.env`	Your secret keys -- LOCAL ONLY -- never goes to GitHub	**NEW**	Phase 0 setup
`.env.example`	Variable names only -- safe to commit	**NEW**	Phase 0, MSG 0-2
`.gitignore`	Protects .env and nova.db	**NEW**	Phase 0, MSG 0-2
`vercel.json`	Routes + security headers	**NEW**	Phase 0, MSG 0-3
`package.json`	npm dependencies	**NEW**	Phase 0, MSG 0-3
`supabase_rls.sql`	Row Level Security -- paste in Supabase	**NEW**	Phase 5, MSG 5-2
`SECURITY_CHECKLIST.md`	Final pre-launch verification list	**NEW**	Phase 5, MSG 5-3

FRONT END

FRONTEND CHANGES -- 4 MESSAGES TO SEND CLAUDE

4 messages total -- fits inside the phases of NOVA_Guide.pdf

[!] Use the same context phrase from the last page of NOVA_Guide.pdf before each message below. Paste context phrase first, then paste the message phrase, then attach the listed files.

PART A -- THINGS YOU DO YOURSELF (for the frontend messages)

1 After MSG 1-5 (checkout.js updated): remove the EmailJS script tag from index.html if you want. It no longer does anything because the code no longer calls it. The script tag is near the top of index.html -- look for emailjs.com in the src attribute.

2 After MSG 3-3 (index.html updated): open your store in the browser and check that products still load. Open browser DevTools (F12) -> Console tab -- there should be no errors. Products now come from /api/products instead of products.js.

3 After MSG 4-3 (reviews.js updated): submit a test review on your store. Go to /admin/reviews -> Pending tab -> approve it -> refresh the store -- it should appear.

4 After MSG 4-4 (engagement.js updated): sign up for the newsletter on the store. Go to /admin/newsletter to see the subscriber appear.

PART B -- MESSAGES TO SEND CLAUDE

MSG 1-5 checkout.js (updated -- replaces EmailJS with fetch to /api/orders)

Claude creates:

`checkout.js`
Updated file. placeOrder() now sends order to /api/orders instead of emailjs.send(). Same form, same validation, same success/error screens.

If new session -- attach:

`checkout.js`
The file being changed -- Claude edits placeOrder() inside it

`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.
The cart items in my checkout.js are stored like this:
item.id -- product id string e.g. "acc-001"
item.productRef -- the full product object (may be null if not found)
item.name -- display name string (already resolved)
item.price -- number
item.qty -- number
item.size -- string
item.color -- string or null
```

```
Use this exact fetch:
const res = await fetch("/api/orders", {
method: "POST",
headers: {"Content-Type": "application/json"},
body: JSON.stringify({
name: name,
address: address,
phone: phone,
lang: currentLang,
cart: cart.map(item => ({
id: item.id,
name: item.productRef ? getName(item.productRef) : item.name,
price: item.price,
qty: item.qty,
size: item.size,
color: item.color
}))
})
})
const data = await res.json()
if (!data.success) throw new Error(data.error || "Order failed")
Keep the same success/error behavior that already exists in the file.
Do not change the form, the validation, the cart logic, or the UI.
Return the full updated checkout.js.
```

MSG 3-3 index.html (updated -- store loads products from /api/products)

Claude creates:

`index.html`

Updated file. init() now calls loadProducts() which fetches from /api/products. Static PRODUCTS array kept as fallback if API fails. Nothing else changes.

If new session -- attach:

`index.html`

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

`api/products/index.js`

Claude needs the GET response shape 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 inline script at the bottom of the file.
Step 1: rename PRODUCTS to PRODUCTS_STATIC
Find: let products = PRODUCTS;
Change the PRODUCTS array declaration name to PRODUCTS_STATIC
Change the assignment to: let products = PRODUCTS_STATIC;
Step 2: add loadProducts() before init():
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
// Re-run loadCart so cart items reconnect to fresh product objects
loadCart()
} catch(e) {
products = PRODUCTS_STATIC // silent fallback
}
renderProducts()
blurUpObserve()
updateCartUI()
updateWishlistUI()
}
Step 3: in init(), find the renderProducts() call.
Replace it with: loadProducts()
Remove blurUpObserve(), updateCartUI(), updateWishlistUI() from init()
because loadProducts() now calls them after the fetch resolves.
IMPORTANT: the API returns products with name and desc as flat strings:
name_en, name_ar, desc_en, desc_ar (separate fields)
My frontend expects: p.name = {en:"...", ar:"..."} and p.desc = {en,ar}
After fetching, convert each product:
product.name = {en: product.name_en, ar: product.name_ar}
product.desc = {en: product.desc_en, ar: product.desc_ar}
product.oldPrice = product.old_price
Also convert colors: each color.name = {en: color.name_en, ar: color.name_ar}
Change ONLY init() and add loadProducts() -- touch nothing else.
Return the complete updated index.html.
```

MSG 4-3 reviews.js (updated -- loads reviews from /api/reviews)

Claude creates:

`reviews.js`

Updated file. renderReviews() now fetches from /api/reviews first.
Falls back to REVIEWS_DATA array if API fails or returns empty.
Handles both API format and static array format.

If new session -- attach:

`reviews.js`

The file being changed -- Claude updates renderReviews() only

`api/reviews/index.js`

Claude needs the GET response format to know the field names

Copy-paste phrase:

```
NOVA Phase 4, Message 3.
# Sharing: reviews.js and api/reviews/index.js
Update reviews.js -- change ONLY the renderReviews() function.
Keep REVIEWS_DATA array in the file -- it is the fallback.
Keep renderStars() unchanged.
Make renderReviews() async and fetch from API at the start:
async function renderReviews() {
// ... keep the heading/sub text update code unchanged ...
let data = REVIEWS_DATA; // start with fallback
try {
const res = await fetch("/api/reviews")
if (res.ok) {
const fetched = await res.json()
if (Array.isArray(fetched) && fetched.length > 0) {
data = fetched
}
}
} catch(e) { /* silent -- use REVIEWS_DATA fallback */ }
// The API returns these fields:
// product_name_en, product_name_ar, stars, text_en, text_ar,
// author, created_at
// REVIEWS_DATA has:
```

```
// product:{en,ar}, stars, text:{en,ar}, name, avatar, date:{en,ar}
// Detect which format and render both:
grid.innerHTML = data.map(r => {
const isApi = r.text_en !== undefined;
const product = isApi
? (currentLang==="ar" ? r.product_name_ar : r.product_name_en)
: (r.product[currentLang] || r.product.en);
const text = isApi
? (currentLang==="ar" ? r.text_ar : r.text_en)
: (r.text[currentLang] || r.text.en);
const author = isApi ? r.author : r.name;
const avatar = isApi
? `https://i.pravatar.cc/150?u=${encodeURIComponent(r.author)}`
: r.avatar;
const date = isApi
? new Date(r.created_at).toLocaleDateString(currentLang==="ar"?"ar-TN":"en-GB",
{year:"numeric",month:"long"})
: (r.date[currentLang] || r.date.en);
// ... use the same HTML template for the card as before ...
}).join("")
}
Return the complete updated reviews.js.
```

MSG 4-4 engagement.js (updated -- newsletter form POSTs to /api/newsletter)

Claude creates:

`engagement.js`

Updated file. Newsletter form submit handler now POSTs to /api/newsletter. Same toast notifications. Nothing else changes.

If new session -- attach:

`engagement.js`

The file being changed -- Claude finds the newsletter submit handler

`api/newsletter/index.js`

Claude needs the POST body format {email, lang} and response {success:true}

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 on the email input form.
It currently either shows a success toast immediately or does nothing.
Replace the submission logic with this fetch call:
e.preventDefault()
const email = /* however the email value is currently read */
if (!email || !email.includes("@")) return
fetch("/api/newsletter", {
method: "POST",
headers: {"Content-Type": "application/json"},
body: JSON.stringify({
email: email,
lang: currentLang
})
})
.then(r => r.json())
.then(data => {
```

```
if (data.success) {
// show the same success toast that already exists in the file
} else {
throw new Error()
}
})
.catch(() => {
// show the same error toast that already exists in the file
})
Keep ALL other engagement.js code unchanged.
-- popups timing
-- sale banner
-- exit intent
-- all other functions
Return the complete updated engagement.js.
```

AFTER ALL 4 FRONTEND MESSAGES -- FINAL CHECKS

1 Commit all 4 updated files to GitHub -> Vercel auto-deploys.

2 Open your store. Products should load from the database. Check DevTools console for errors.

3 Place a test order. Check /admin/orders -- it should appear. Check your email.

4 Submit a test review. Approve it in /admin/reviews. Refresh store -- it should appear.

5 Sign up for newsletter. Check /admin/newsletter -- subscriber should appear.

ALL FRONTEND MESSAGES SUMMARY

Msg	Builds	Attach if new session	Do after
1-5	checkout.js (updated)	checkout.js + api/orders/index.js	Remove EmailJS script tag from index.html
3-3	index.html (updated)	index.html + api/products/index.js	Check store loads products, check console for errors
4-3	reviews.js (updated)	reviews.js + api/reviews/index.js	Submit test review, approve in admin, verify on store
4-4	engagement.js (updated)	engagement.js + api/newsletter/index.js	Sign up for newsletter, check /admin/newsletter

NOVA -- Frontend Changes Guide

4 files change | 14 files untouched | Use with NOVA_Guide.pdf