

FlowerWax Technical Handover

Clear operating guidance for the new owner and technical operator

1. Product and system overview

FlowerWax is a responsive wholesale marketplace for botanical beauty ingredients. The transferred application package includes a React single-page web application, Convex backend functions and data storage, Clerk user authentication, Stripe payments, supplier storefronts, product management, order handling, verified reviews, contact handling, and a separate owner administration area.

The product is a starter asset. Configure buyer-owned service accounts and complete payment, security, and deployment checks before public launch.

2. Main application areas and roles

Public area

Public routes provide the home page, marketplace, pricing, about, contact, legal information, product details, and published supplier storefronts. Public data reads exclude disabled and unpublished suppliers.

Customer area

Clerk signs customers in. A customer record is stored in the shared users table with the consumer role. Customers can maintain a same-supplier cart, pay, view purchases, and submit one verified review for each purchased product.

Supplier area

Suppliers use the same authentication flow with the provider role. They can pay a configured one-time setup fee, connect a Standard Stripe account, update storefront details, create and manage products, publish a store, view sales, and update fulfilment status.

Owner administration area

The private administration route uses a separate server-checked credential session. It provides user, product, order, settings, and contact-message management. Rotate the initial transfer credential before launch and do not reuse it for any third-party service.

3. Buyer-safe architecture summary

The browser application is built and bundled by Vite. React Router handles public and protected routes. Clerk provides browser identity, and the Convex React client provides live queries and mutations.

Convex is the application backend. Its queries, mutations, actions, HTTP routes, storage, and scheduled functions enforce permissions and carry out data operations. Stripe secrets stay in the Convex environment. Product checkout reloads prices from Convex, checks that all cart items belong to one supplier, and creates a Payment Intent with the supplier's connected account as the payout destination.

Stripe sends signed events to the Convex HTTP webhook. The handler checks the signature, site identifier, event replay state, and payment metadata before creating an order. This avoids trusting browser totals and reduces duplicate-order risk.

The frontend builds to static files. A normal shared host that supports HTTPS, uploaded static files, and SPA route fallback is enough for the frontend. Convex remains the managed backend service.

4. Technology stack

- React 19 and TypeScript for the web application.
- Vite 6 for development and production bundling.
- React Router for routing.
- Tailwind CSS for styling.
- Framer Motion for interface motion.
- Clerk for customer and supplier authentication.
- Convex for data, backend functions, HTTP routes, and file storage.
- Stripe and Stripe Connect for setup payments, customer checkout, and supplier payouts.
- GitHub Actions and FTPS configuration for the existing static hosting workflow.
- pnpm 10 as the declared package manager.

5. Project structure

- `App.tsx`: Main route tree and protected route handling.
- `AppConvex.tsx`: Clerk and Convex provider setup plus owner route selection.
- `src/pages/`: Public pages, onboarding, dashboards, storefronts, authentication, legal pages, and administration screens.
- `src/components/`: Navigation, marketplace cards, cart, checkout, pricing, FAQ, and shared interface components.
- `src/context/AppContext.tsx`: Client access to the signed-in user, products, cart, orders, settings, and mutations.
- `src/lib/`: Shared types, utility functions, data used for demonstrations, and image references.
- `convex/schema.ts`: Authoritative application data model.
- `convex/users.ts`: User sync, roles, storefront settings, Stripe state, and platform settings.
- `convex/products.ts`: Public catalogue reads and supplier-owned product changes.
- `convex/orders.ts`: Customer purchases, supplier sales, fulfilment status, and administration totals.
- `convex/stripe.ts`: Setup fee checkout, Stripe Connect onboarding, and product Payment Intents.

- `convex/webhooks.ts`: Signed Stripe event processing and idempotent order creation.
- `convex/reviews.ts`: Purchase-backed review rules.
- `convex/contact.ts`: Public contact submission and validation.
- `convex/admin.ts`: Private owner sessions and administration operations.
- `convex/files.ts`: Authenticated file upload URLs.
- `convex/http.ts`: Health and Stripe webhook routes.
- `scripts/audit-srs.mjs`: Focused product and secret-hygiene checks.
- `public/`: Static public assets, robots file, sitemap, and host configuration copied during build.

6. Data model

The current schema uses these tables:

users

Stores the Clerk identity, name, email, role, avatar, supplier storefront fields, Stripe account state, setup-payment state, publish state, and disabled state. Customer and supplier roles share this table.

Important indexes support lookup by Clerk ID, email, and public store address.

products

Stores the product name, supplier identity, supplier name snapshot, price, selling unit, image reference, category, description, and average rating. Products are indexed by supplier and category.

orders

Stores one purchased product snapshot per record, the customer and supplier identities, amount, optional quantity and unit price, date, fulfilment status, shipping address, Payment Intent ID, and Stripe event ID. Indexes support customer, supplier, product, and payment lookups.

reviews

Stores the product, customer identity, customer name, rating, comment, and date. The product and customer index enforces one review per customer and product.

platformSettings

Stores the public application name, optional logo, one-time supplier setup fee, and optional Stripe price reference.

adminCredentials and adminSessions

Store the hashed private owner credential and hashed administration sessions. The browser does not authorize itself.

contactMessages

Stores validated contact form submissions and their new, read, or resolved status.

stripeEvents

Stores processed Stripe event IDs and timestamps for replay protection.

testimonials

Stores optional public testimonial content. Do not publish unverified testimonials or success claims.

The schema file is the source of truth. Review it before adding fields or changing validation.

7. Required buyer-owned services

Create buyer-owned accounts for:

- Clerk authentication.
- Convex backend and storage.
- Stripe and Stripe Connect.
- The domain registrar and DNS service.
- HTTPS static hosting with SPA fallback support.
- A private source-control account and deployment secrets if automated deployment is retained.

Do not copy seller keys into a public document or browser variable. Rotate every transferred credential.

8. Environment configuration

Create `.env.local` in the project folder for public frontend values:

```
VITE_CONVEX_URL=https://YOUR_CONVEX_DEPLOYMENT.convex.cloud
VITE_CLERK_PUBLISHABLE_KEY=YOUR_CLERK_PUBLISHABLE_KEY
VITE_STRIPE_PUBLISHABLE_KEY=YOUR_STRIPE_PUBLISHABLE_KEY
VITE_SITE_URL=https://YOUR_SITE_URL
```

Set private values in the buyer-owned Convex environment. Use the names expected by the source package, including:

```
STRIPE_SECRET_KEY=YOUR_SERVER_ONLY_STRIPE_SECRET
STRIPE_WEBHOOK_SECRET=YOUR_SERVER_ONLY_WEBHOOK_SECRET
SITE_URL=https://YOUR_SITE_URL
SITE_ID=YOUR_LOWERCASE_SITE_ID
```

Clerk issuer or signing configuration and any bounded seed access value must also be set as required by the included local setup notes. Keep all secret values server-only. Never prefix a secret with `VITE_` because Vite exposes those variables to the browser.

9. Local setup

Use the package manager declared by the project:

```
pnpm install
pnpm exec convex dev --once
pnpm dev
```

Run `pnpm install` only when dependencies are missing or the package manifest or lockfile changed.

For focused verification:

```
pnpm typecheck
pnpm audit:srs
```

Run the production build once near release:

```
pnpm build
```

The output is written to `dist/`.

10. Deployment sequence

1. Create the buyer-owned Clerk, Convex, and Stripe projects.
2. Configure Clerk to issue identity accepted by the Convex deployment.
3. Set the private Convex environment values.
4. Deploy the Convex schema and functions.
5. Add the public frontend values and run the focused checks.
6. Build the frontend once with `pnpm build`.
7. Upload the contents of `dist/` to the shared hosting web root.
8. Confirm that the included SPA fallback is active so inner routes return the application.
9. Point the domain to the host and enable HTTPS.
10. Configure the Stripe webhook and complete test-mode acceptance checks.
11. Move to live Stripe mode only after the test flow passes and the business policies are ready.

The included automation can deploy the static files by FTPS after the required encrypted values are added to the buyer's private source-control settings. Manual static upload is also possible.

11. Stripe setup and payment flow

Supplier setup fee

The owner sets `platformFee` from the private administration settings. When the value is greater than zero, a supplier checkout session charges the one-time setup fee. A successful signed event marks the supplier setup as paid.

Supplier connection

The supplier starts Stripe Connect onboarding from the dashboard. The backend creates or resumes a Standard connected account and stores its account ID. The platform should publish the supplier only after Stripe reports that account details, charges, and payouts are ready.

Product checkout

The customer submits product IDs, quantities, and a shipping address. The server then:

1. Validates cart size and quantity limits.
2. Reloads current products and prices.
3. Rejects products from more than one supplier.
4. Checks that the supplier is published and payout-ready.
5. Creates a Stripe Payment Intent with the supplier as the transfer destination.
6. Sends only the client secret and checked amount back to the browser.

The current payment flow sends product money to the supplier. It does not include an owner application fee on each product sale.

Webhook checks

Subscribe the configured webhook endpoint to the event types required by the included Stripe handler. Confirm signature verification, correct site isolation, replay protection, successful order creation, and failed-payment handling. Never accept an unsigned test request as proof of the live flow.

12. Authentication and authorization

Clerk handles customer and supplier sign-in. Convex functions check the authenticated identity and role again on the server. Supplier product changes are limited to the supplier's own records. Customers can review only products with a matching purchase. Public marketplace reads exclude disabled or unpublished suppliers.

The private owner area uses a separate session token. Change its username and password immediately after deployment. Remove old sessions during credential rotation if operational policy requires it.

13. Security and credential rotation

- Replace all transferred service keys and passwords.
- Keep Stripe secrets, webhook secrets, deployment keys, hosting passwords, and owner passwords out of source files.
- Use only buyer-owned accounts after handover.
- Restrict access to production service dashboards.
- Enable multi-factor authentication where supported.
- Confirm HTTPS before entering customer or payment information.
- Keep server authorization checks in place when changing the interface.
- Preserve order history when disabling users or removing catalogue content.
- Review dependency updates and security notices on a regular schedule.

14. Functional verification checklist

Public and account flows

- Home, marketplace, pricing, about, contact, legal, and storefront routes load over HTTPS.
- Clerk sign-in works for a new customer and a new supplier.
- Role selection and onboarding reject invalid transitions.
- Disabled and unpublished suppliers do not appear publicly.

Supplier and catalogue flows

- A supplier can complete the setup fee flow when the fee is enabled.
- Stripe Connect onboarding returns to the correct website.
- A supplier can update the store, upload an image, create a product, edit it, and publish the store.
- One supplier cannot change another supplier's products.

Checkout and order flows

- Mixed-supplier carts are blocked.
- Cart totals come from current server data.
- A successful signed Stripe event creates the expected order once.
- A repeated event does not create a duplicate order.
- A failed payment creates no completed order.
- Customers see purchases and suppliers see sales.
- Suppliers can move an order from pending to shipped to completed, but cannot reopen a completed order.
- Only a customer with a matching purchase can create one review for a product.

Owner flows

- The private owner sign-in works after credential rotation.
- Users, products, orders, settings, and contact messages load.
- The owner can update the public name, logo, and supplier setup fee.
- Disabling a user preserves related products and order history.

15. Backup, monitoring, and maintenance

- Use Convex export or backup tools on a regular schedule suitable for order volume.
- Keep a private copy of environment names and recovery steps without storing secret values in the document.
- Monitor Stripe webhook delivery and failed events.
- Monitor the public website, the Convex health route, authentication, and one inner SPA route.
- Review contact messages and failed customer payments.
- Keep domain, hosting, Clerk, Convex, and Stripe billing current.

- Test restoration and credential rotation before they are urgently needed.

16. Product and support boundaries

FlowerWax has no represented production traffic, revenue, customer history, or uptime record. Buyer-owned service accounts, live payment activation, real supplier content, operating policies, and final legal review are required before launch.

Codeblix provides up to 30 days of technical transition support. The support covers setup guidance, deployment questions, and explanation of the current application package. It does not include unlimited development, ongoing operations, supplier verification, marketing, legal work, or third-party fees unless separately agreed.