

RecipeGPT Technical Documentation

Architecture, deployment, security, and maintenance for the transferred source.

CODEBLIX TRANSFER EDITION | 22 AUGUST 2026

Architecture overview

RecipeGPT is a React single-page application with a Convex backend. Vite builds the frontend for static Apache-compatible hosting. Clerk handles user identity. Convex provides the database, typed queries, mutations, actions, file storage, and realtime updates. Stripe handles subscriptions and the billing portal. Gemini is called from a Convex action and is never called directly from the browser.

The source repository is `github.com/codeblix-ltd/FLIPPA-Recipe-GPT`. The project is designed to use a dedicated buyer-owned Convex deployment. Do not reuse another product's deployment.

Technology stack

- React 19 and React DOM
- TypeScript
- Vite 8
- React Router
- Tailwind CSS 4 plus product-specific CSS
- Clerk React authentication
- Convex database, functions, actions, storage, and generated types
- Stripe Checkout, Billing Portal, products, prices, subscriptions, and signed webhook handling
- Gemini 2.5 Flash for structured rescue meal generation
- Lucide React icons
- Faker for deterministic seed data
- Apache static hosting with SPA fallback through `public/.htaccess`

Repository structure

```

FLIPPA-Recipe-GPT/
|-- src/
|   |-- components/      Brand, layout, route guards, and shared UI
|   |-- data/            Product plans, public recipes, navigation, starter pantry
|   |-- pages/public/    Homepage, marketing, pricing, recipes, legal
|   |-- pages/app/       Dashboard, pantry, rescue, shopping, planner, settings
|   |-- App.tsx          Browser routes and user synchronisation
|   |-- index.css        Product design system and responsive UI
|   |-- main.tsx         React entry point
|-- convex/
|   |-- schema.ts        Convex data model and indexes
|   |-- users.ts         User synchronisation
|   |-- households.ts    Household creation and membership
|   |-- pantry.ts        Pantry CRUD and image storage IDs
|   |-- ai.ts            Structured rescue generation and fallback
|   |-- aiUsage.ts       Generation usage records
|   |-- kitchen.ts       Overview, cook completion, and leftovers
|   |-- planner.ts       Weekly meal plans
|   |-- shopping.ts      Shared shopping list and pantry return flow
|   |-- billing.ts       Checkout and customer portal actions
|   |-- http.ts          Stripe webhook route
|   |-- stripeEvents.ts  Idempotent webhook event handling
|   |-- adminAuth.ts     Hashed admin credentials and sessions
|   |-- seed.ts          Idempotent plans, recipes, ingredients, and demo data
|-- public/              Logos, favicon, and Apache fallback rules
|-- srs.txt              Product requirements and architecture plan
|-- SRS-COMPLETION-MATRIX.md  Implemented feature map
|-- INSTALLATION.md      Environment and deployment instructions
|-- package.json         Scripts and dependency contract
|-- vite.config.ts       Vite configuration

```

Route map

Public routes include `/`, `/how-it-works`, `/pricing`, `/recipes`, `/recipes/:slug`, `/about`, `/privacy`, and `/terms`. The authentication route is `/auth`.

Protected routes include `/onboarding`, `/app`, `/app/pantry`, `/app/rescue`, `/app/planner`, `/app/shopping`, `/app/recipes`, `/app/leftovers`, `/app/household`, `/app/settings`, `/app/billing`, and `/app/cook`.

Convex data model

The schema includes these main areas:

Identity and households

- `users` stores Clerk identity, email, name, plan code, subscription status, preferences, and onboarding state.
- `households` stores the household name, owner identity, invite code, and timestamps.
- `householdMembers` links Clerk identities to households with an owner or member role.

Pantry and cooking

- `pantryItems` stores ingredient name, normalized name, quantity, unit, category, expiry timestamp, optional storage location, and optional Convex storage ID.

- `recipes` stores structured recipe content, ingredients, steps, tags, rescue score, optional images, public state, and status.
- `cookingSessions` records completed cooking sessions and the number of items rescued.
- `leftovers` stores household, recipe name, servings, use-by time, and creation time.
- `aiUsage` records rescue generation model, input item count, success state, household, and timestamp.

Planning and commerce

- `mealPlans` stores week, day, meal type, recipe, servings, and status.
- `shoppingItems` stores shared items, quantities, categories, source, purchase state, and timestamps.
- `plans` stores Free, Plus, and Family price data, annual price data, features, active state, and optional Stripe IDs.
- `subscriptions` stores buyer-owned Stripe customer and subscription references, plan code, status, and period end.
- `webhookEvents` stores Stripe event IDs so webhook processing is idempotent.

Administration

- `adminCredentials` stores a username and hashed password value.
- `adminSessions` stores a hashed session token and expiry.

Important backend flows

User synchronisation

`src/App.tsx` calls `users.sync` after Clerk loads. The mutation creates or updates the Convex user record. It does not store a private Clerk secret in the browser.

Onboarding

`households.completeOnboarding` creates a household, inserts the starter pantry, links the owner, and marks onboarding complete. The mutation is protected by the current authenticated identity.

Rescue generation

`ai.generateRescueMeals` loads the user's pantry, constructs a structured prompt, calls Gemini from the Convex action when configured, validates that three results were returned, and records the outcome through `aiUsage.record`. A deterministic fallback is used when there is no key, no pantry, or a failed request.

Cooking and leftovers

`kitchen.completeCooking` checks the household, reduces a bounded set of pantry quantities, records a cooking session, and creates a leftover record when the submitted serving count is greater than zero. The deduction rule is intentionally simple and should be reviewed before a buyer relies on exact stock levels.

Shopping return flow

`shopping.togglePurchased` changes purchase state. When an item changes to purchased, the current mutation inserts a corresponding pantry item with a seven-day default expiry window. This is a starter rule and should be made configurable for production use.

Billing

`billing.createCheckout` checks identity and plan state, creates missing Stripe product and monthly price mappings, creates a Checkout Session, and passes app and plan metadata to the session and subscription. `billing.createPortal` opens the Stripe customer portal. `/stripe/webhook` verifies the signature before `stripeEvents.process` stores the event ID and applies subscription state.

Environment variables

Public Vite values

- `VITE_CONVEX_URL`
- `VITE_CONVEX_SITE_URL`
- `VITE_CLERK_PUBLISHABLE_KEY`
- `VITE_STRIPE_PUBLISHABLE_KEY`
- `VITE_SITE_URL`

Convex server values

- `CLERK_JWT_ISSUER_DOMAIN`
- `GEMINI_API_KEY`
- `STRIPE_SECRET_KEY`
- `STRIPE_WEBHOOK_SECRET`
- `SITE_URL`
- `APP_ID`
- `APP_NAME`
- `CONVEX_SITE_ID`

Only public browser values may use the `VITE_` prefix. Never put Gemini, Stripe secret, webhook, Convex deployment, or Clerk server values into the frontend bundle, this document, the sales listing, screenshots, or Git.

Local setup

The buyer should use `pnpm` and a supported Node.js runtime. The normal setup is:

```
pnpm install
pnpm exec convex dev --once
pnpm seed
pnpm check
pnpm build
```

This documentation was prepared without installing dependencies or running the UI. Before accepting real users, run the commands against the buyer's own deployment and environment.

Production deployment

Convex backend

1. Create or select the buyer-owned production deployment.
2. Set the server-only environment variables.
3. Deploy the Convex functions.
4. Run the idempotent seed against the intended deployment.
5. Configure Stripe to send signed checkout and subscription events to the deployed `/stripe/webhook` route.

Static frontend

1. Set the public Vite environment values.
2. Run one production build.
3. Upload the `dist` directory, including `.htaccess`, to Apache hosting.
4. Confirm direct navigation for nested React routes.
5. Confirm that the public domain uses the buyer's Convex deployment and Clerk application.

The source does not prove that a production frontend, Convex deployment, Stripe account, Clerk account, or Gemini account is currently live. Those services are part of the buyer handover and must be confirmed independently.

Security model

- Protected Convex functions require an authenticated Clerk identity.
- Household queries and mutations check household membership before reading or changing records.
- Server-only keys are read from Convex environment variables.
- Stripe webhook requests require signature verification and event ID checks.
- Admin credentials are hashed and admin sessions are stored as hashes with expiry.
- Convex storage IDs are used for pantry images instead of base64 fields.
- The product copy warns users that recipes are assistance, not medical advice.
- The buyer should rotate all seed, Clerk, Convex, Stripe, Gemini, hosting, and repository credentials during handover.

Verification checklist

- Public homepage, recipes, pricing, and legal pages load.
- Clerk sign-up and sign-in complete.
- Onboarding creates one household and the starter pantry.
- Pantry add and remove operations work.
- Rescue returns three structured results with and without Gemini configured.
- Cook mode records a session and optional leftovers.
- Shopping items can be added, purchased, and returned to pantry.

- Weekly plan and household pages load with the correct household scope.
- Free and paid checkout flows work in Stripe test mode.
- Invalid Stripe signatures are rejected.
- Admin sessions expire and seed credentials are rotated.
- The production build and nested route fallback work on buyer-owned hosting.

Known MVP boundaries

- The current pantry deduction logic is intentionally approximate.
- The current household invite mutation returns a pending invite shape and should be connected to a real delivery and acceptance flow before launch.
- Photo pantry storage is implemented at the Convex schema and upload-flow level. Expanded image recognition should be tested before it is marketed as a guaranteed workflow.
- The product has no represented live customer, revenue, traffic, or retention dataset.
- The buyer must review the RecipeGPT name and GPT trademark considerations before relaunch.

Transfer checklist

1. Transfer the domain and confirm DNS ownership.
2. Transfer the GitHub repository and confirm the default branch.
3. Create buyer-owned Convex, Clerk, Stripe, Gemini, and hosting accounts.
4. Set server and browser environment variables without committing secrets.
5. Deploy and seed the backend.
6. Build and publish the static frontend.
7. Run the verification checklist.
8. Rotate the seeded admin credential and all temporary service access.
9. Confirm the buyer has the source, documents, domain, and deployment notes.
10. Remove seller access after the buyer confirms receipt.