

Macro-Manage Technical Handover Guide

Buyer technical guide | Macro-Manage.com

1. System overview

Macro-Manage is a multi-organization fee-management web application for schools, clubs, tutors, gyms, academies, and membership groups. It includes:

- A public marketing website.
- Customer registration and sign-in.
- A protected organization workspace.
- Roster and recurring-fee management.
- Collection reports and CSV export.
- AI-assisted roster import.
- Recurring SaaS subscription checkout and billing management.
- A separate private platform administration area.

Each signed-in customer belongs to an organization. Roster records, payments, settings, imports, and reports are scoped to that organization. The platform owner sells application access to organizations. The application records the fees those organizations collect, but it does not process student or member fee payments.

2. Main application areas and roles

Public website

Public routes cover the home page, features, industries, pricing, about, privacy, terms, contact, and customer authentication. Unknown routes show an application not-found page.

Organization workspace

Protected routes cover overview, roster, fee grid, analytics, image import, subscription billing, and organization settings.

Organization roles

- Owner: full organization management rights.
- Manager: daily roster, payment, import, and settings actions.
- Viewer: read-only access enforced by protected server mutations.

Platform administrator

The private administration area uses a separate session. It provides platform totals, customer account status, organization visibility, access suspension or activation, and administrator credential changes. Administrator sessions expire after 12 hours.

3. Architecture

The frontend is a React single-page application built by Vite. React Router handles public, customer, dashboard, administration, and fallback routes. Clerk provides customer identity and browser sessions. Convex provides the data model, authorization-aware server functions, file storage, actions, and HTTP endpoints. Stripe provides hosted subscription checkout, recurring billing state, signed webhooks, and the customer billing portal. Gemini provides image-to-roster extraction when enabled.

The production frontend is static and can be hosted on shared Apache hosting with HTTPS and single-page application rewrite support. Convex remains the hosted application backend. Buyer-owned accounts are required for all external services.

4. Technology stack

- React 19 for the user interface.
- TypeScript for application and server code.
- Vite 6 for development and production builds.
- React Router 7 for routing.
- Clerk for customer authentication.
- Convex for data, server functions, file storage, and HTTP actions.
- Stripe for SaaS subscription billing.
- Gemini for optional image-to-roster extraction.
- pnpm 10 for package management.
- Node.js 24 or a compatible supported release.
- Apache rewrite rules for static hosting routes.

5. Project structure

<code>components/</code>	Public marketing components
<code>src/components/</code>	Shared authentication and application components
<code>src/pages/</code>	Public, authentication, admin, and information pages
<code>src/pages/dashboard/</code>	Organization workspace pages
<code>convex/</code>	Schema, server functions, actions, billing, and HTTP routes
<code>public/</code>	Images, icons, manifest, service worker, and rewrite rules
<code>.github/workflows/</code>	Optional automated deployment workflow
<code>dist/</code>	Generated production output

Edit source files, then create a new `dist/` build. Do not treat generated output as the editable source.

6. Data model

users

Stores the external identity ID, email, display name, image, account status, customer billing ID, current plan, subscription status, and timestamps. Indexed lookups support identity, email, and billing customer ID.

organizations

Stores organization name, URL-safe slug, industry, owner identity, and timestamps. Indexed by slug and owner.

memberships

Links customer identities to organizations with owner, manager, or viewer access. Indexed by user, organization, and the combined organization-user pair.

students

Stores organization-scoped student or member records, contact details, custom fields, enrollment years, active status, optional avatar storage ID, and timestamps.

transactions

Stores organization, roster record, month, year, amount, payment method, payment date, note, creating user, and creation time. A month may contain several payments.

settings

Stores each organization's standard monthly fee, currency, color settings, custom roster fields, reminder preference, and update time.

imports

Stores uploaded source file ID, filename, processing status, created row count, error text, creator, and processing times.

plans

Stores plan code, public name, price, billing interval, description, features, highlighted state, payment product and price IDs, active state, and update time.

subscriptions

Stores the organization, plan, payment customer, subscription, checkout session, status, renewal date, and update time.

webhookEvents

Stores processed Stripe event IDs and types. The event ID index prevents duplicate webhook processing.

adminCredentials and adminSessions

Store the private platform administrator password hash, salt, change time, session token, username, expiry, and creation time. Plain passwords are not stored.

contactRequests

Stores public contact form messages with new or resolved status.

auditLogs

Stores selected organization or platform activity with actor, action, entity type, optional entity ID, metadata, and creation time.

7. Required buyer-owned services

Create separate buyer-owned development and production accounts where possible:

- Clerk application for customer sign-in.
- Convex deployments for data, server functions, and file storage.
- Stripe account for paid SaaS plans.
- Gemini API access for image import.
- HTTPS hosting for the static frontend.
- DNS access for Macro-Manage.com.

Do not reuse seller credentials. Replace every temporary, shared, test, bootstrap, and transferred value before opening customer registration.

8. Browser configuration

Copy the provided environment example to `.env.local` and use buyer-owned values:

```
VITE_CONVEX_URL=https://YOUR_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
```

Only browser-safe values may use the `VITE_` prefix. Never place a private key or server secret in a `VITE_` variable because it becomes part of the browser build.

9. Server configuration

Set these values in both intended Convex environments:

```
CLERK_SIGNING_KEY=https://YOUR_CLERK_ISSUER_DOMAIN
STRIPE_SECRET_KEY=YOUR_STRIPE_SECRET_KEY
STRIPE_WEBHOOK_SECRET=YOUR_STRIPE_WEBHOOK_SECRET
GEMINI_API_KEY=YOUR_GEMINI_API_KEY
GEMINI_MODEL=YOUR_SUPPORTED_MODEL
SITE_URL=https://YOUR_SITE_URL
APP_ID=YOUR_APP_ID
APP_NAME=Macro-Manage
CONVEX_SITE_ID=YOUR_CONVEX_DEPLOYMENT_NAME
```

The current `CLERK_SIGNING_KEY` variable is used as the Clerk issuer domain in the authentication provider configuration. Supply the buyer's issuer domain. Keep payment, webhook, and AI secrets in the server environment only.

10. Installation and local development

Requirements are Node.js 24 and pnpm 10.

```
pnpm install
pnpm convex dev
pnpm dev
```

Use test service accounts and fictional data during local work. Do not point normal development activity at live customer data.

Before a release, run:

```
pnpm typecheck
pnpm convex dev --once --typecheck enable
pnpm build
```

The generated frontend is written to `dist/`.

11. Customer authentication setup

- 1 Create a buyer-owned Clerk application.
- 2 Add the production domain and local development origin to allowed origins and redirects.
- 3 Create the JWT template required by Convex and name it `convex`.
- 4 Put the publishable key in the browser configuration.
- 5 Put the issuer domain in the server-side `CLERK_SIGNING_KEY` value.
- 6 Test sign-up, sign-in, sign-out, protected-route redirects, user synchronization, and automatic workspace creation.

If sign-in works but data does not load, check the JWT template name, issuer domain, Convex authentication provider, user status, and active organization membership.

12. Convex setup

- 1 Create buyer-owned development and production deployments.
- 2 Set server environment values in both deployments.
- 3 Deploy the schema and server functions.
- 4 Confirm a new customer receives a user, organization, owner membership, and default settings.
- 5 Confirm one organization cannot read or change another organization's data.
- 6 Confirm viewer accounts cannot run write actions.

Current operational limits include up to 1,000 roster records in a roster read, 5,000 yearly transactions in a transaction read, 25 recent import records, and 25 extracted rows per image import. Review these limits before onboarding larger customers.

13. Stripe subscription setup

- 1 Create a buyer-owned Stripe account and begin in test mode.
- 2 Set `STRIPE_SECRET_KEY`, `SITE_URL`, and `CONVEX_SITE_ID` in the server environment.
- 3 Create a webhook endpoint:

```
https://YOUR_CONVEX_SITE_ID.convex.site/stripe/webhook
```

- 4 Subscribe it to `checkout.session.completed`, `customer.subscription.updated`, and `customer.subscription.deleted`.
- 5 Store the endpoint signing secret in `STRIPE_WEBHOOK_SECRET`.
- 6 Review every active plan's name, price, interval, description, and features.
- 7 Complete a test checkout and confirm the return URL, customer record, plan, and subscription status.

- 8 Open the billing portal and test a subscription change or cancellation.
- 9 Review Stripe live-mode requirements before replacing test values.

For a paid plan without payment product IDs, the server can create the Stripe product and recurring price on first checkout. Confirm the stored plan values before that first checkout.

14. AI import setup

- 1 Create a buyer-owned Gemini API key.
- 2 Set `GEMINI_API_KEY` and a supported `GEMINI_MODEL` in the server environment.
- 3 Upload a clear image with fictional roster data.
- 4 Confirm the original file is stored and processing completes.
- 5 Confirm no more than 25 rows are created.
- 6 Compare every created record with the source image.

Accepted image formats are JPG, PNG, WEBP, HEIC, and AVIF. The processing limit is 18 MB. Keep the feature disabled if no buyer-owned key is configured.

15. Plans, demo data, and administrator setup

The included seed routine is manual and idempotent. It can create plans, sample organizations, sample roster records, sample transactions, and an initial administrator record.

If seeding is needed, run it once in the intended environment:

```
pnpm convex run --prod seed:run
```

Before public launch:

- 1 Replace the administrator username and password.
- 2 Remove all sample people, organizations, payments, images, and demo account states.
- 3 Review plan prices, intervals, limits, and public wording.
- 4 Confirm no demo subscription is presented as a real customer or revenue record.

Do not place seed commands in automatic deployment.

16. Production deployment

- 1 Configure buyer-owned production services.
- 2 Deploy the Convex schema and server functions.
- 3 Prepare plans and private administrator access.
- 4 Run the type checks.
- 5 Run one production build.
- 6 Upload the contents of `dist/` to the HTTPS document root.
- 7 Enable single-page application fallback so unknown routes serve `index.html`.

- 8 Point Macro-Manage.com to the hosting account.
- 9 Confirm the root domain and chosen www behavior.
- 10 Verify customer sign-in, data access, image upload, payment checkout, webhook delivery, and billing portal.

The static frontend can run on suitable shared Apache hosting. The included rewrite file supports application routes when URL rewriting is available. Convex remains the hosted backend and must be deployed separately.

The service worker registers in production and caches the application shell. Change its cache version when the offline shell changes so older cached files are removed.

17. Security and credential rotation

- Replace every seller, demo, shared, temporary, and bootstrap credential.
- Use buyer-owned accounts with multi-factor authentication.
- Keep private values in server-side environments.
- Keep only publishable values in the browser build.
- Change the private administrator credentials before launch.
- Review the 12-hour administrator session limit.
- Confirm organization data remains isolated through memberships.
- Confirm viewers cannot create, update, archive, import, or record payments.
- Treat uploaded roster images as personal data.
- Define retention and deletion rules for people, payments, images, contacts, billing events, and audit records.
- Complete privacy, consent, refund, tax, and data-protection review for the target market.

18. Functional verification checklist

Public and customer access

- Public pages and direct links load over HTTPS.
- Unknown routes show the application not-found page.
- Contact form validation and storage work.
- Sign-up, sign-in, sign-out, and redirect behavior work.
- New customers receive an organization workspace.

Roster and fees

- Create, edit, search, archive, and restore a roster record.
- Save organization name, industry, fee, currency, and custom fields.
- Record a full payment and confirm paid status.
- Record two partial payments and confirm the combined total.
- Delete a payment and confirm the grid and reports update.
- Change year, search the grid, and export CSV.

- Compare dashboard and analytics totals with test transactions.

Access control

- One organization cannot access another organization's data.
- A viewer cannot perform management actions.
- Suspended users cannot access the protected workspace.
- Reactivation restores access.

Import and billing

- Accepted image formats and size limits work.
- Import states and errors are clear.
- Extracted records match the source image.
- Paid checkout opens in test mode.
- Signed webhook events update subscription state once.
- Duplicate events do not create duplicate processing.
- The customer billing portal opens.

Administration and hosting

- Invalid administrator access is rejected.
- Administrator credentials can be changed.
- Old credentials stop working.
- Direct dashboard URLs return the application shell.
- Manifest, icons, and service worker load correctly.

19. Backup, monitoring, and maintenance

The current package does not include an automatic backup scheduler. Configure service-level backups or exports for application data and uploaded files. Document and test a restore procedure before onboarding real customers.

Monitor:

- Convex function errors, data growth, and storage use.
- Stripe webhook failures, disputes, subscription status, and portal access.
- Gemini quota, model changes, and import errors.
- Hosting availability, HTTPS, DNS, and domain renewal.
- Open contact requests and suspended customer accounts.
- Dependency security updates in a test environment.

Keep a written record of plan changes, environment changes, releases, and incident actions.

20. Current boundaries and support

- There is no verified revenue, traffic, or established customer base.
- Buyer-owned services and live testing are required.
- The contact form stores messages but does not send email.
- The reminder setting does not send automatic reminders by itself.
- AI extraction needs operator review.
- The customer interface does not include a complete team invitation screen.
- The buyer owns payment, tax, legal, privacy, and customer-support decisions.
- Codeblix provides up to 30 days of reasonable handover support for setup questions.
- Support does not include third-party fees, ongoing operation, marketing, legal advice, unlimited custom work, or guaranteed results.