

KillerBug Technical Handover Guide

Setup, deployment, security, and maintenance

1. Product and system overview

KillerBug is a two-sided pest-control marketplace. The browser application serves public marketing pages, authentication, role selection, provider storefronts, homeowner and provider dashboards, and a separate owner administration area.

The browser application is a React single-page application built with TypeScript and Vite. Convex provides application data, server functions, authorization, file storage, scheduled work, and HTTP endpoints. Clerk provides customer and provider identity. Stripe provides provider subscriptions, customer service payments, Stripe Connect payout onboarding, and signed webhook events.

The frontend can be served as static files on shared hosting with an Apache fallback. The Convex backend is deployed separately. All production services must be owned or controlled by the buyer.

2. Main application areas and roles

Public area

Public routes cover the home page, services, company information, service areas, pricing, professional directory, contact form, terms, privacy, and public provider profiles.

Homeowner area

Authenticated homeowners can choose a provider, select one of that provider's saved services, create a request with optional photos, pay after acceptance, monitor progress, review history, and add a review after completion.

Provider area

Authenticated providers can manage profile details, service areas, services, fixed prices, proof documents, direct requests, job progress, notifications, history, and payout onboarding.

Owner administration area

The separate administration area handles users, provider review, contact messages, platform branding, provider billing settings, and admin credential rotation. Admin access does not use the customer and provider sign-in flow.

3. Buyer-safe architecture summary

The main request path is:

1. The browser signs a homeowner or provider in through Clerk.
2. Convex verifies the Clerk identity and reads the stored marketplace role.
3. The browser calls Convex queries, mutations, and actions for marketplace work.
4. Convex checks identity, ownership, provider selection, saved service price, and job state.

5. Stripe handles subscriptions, service payments, Connect onboarding, and signed webhook events.
6. Convex stores the resulting application state and sends it back to the browser.

Sensitive payment decisions remain on the server. The customer-facing amount is derived from the selected provider's saved service price. A pending order is recorded before payment confirmation. Stripe metadata and signed events are checked before payment or subscription state is finalized.

4. Technology stack

- React 19 and React DOM for the browser application.
- TypeScript for typed frontend and backend code.
- Vite for development and static production builds.
- Tailwind CSS 4 for styling.
- Framer Motion for interface motion.
- React Router for browser routing.
- Clerk for homeowner and provider authentication.
- Convex for data, functions, authorization, HTTP routes, scheduled work, and file storage.
- Stripe for provider subscriptions, service payments, Connect payouts, and webhooks.
- pnpm 10 for dependency management.

5. Project structure

Useful relative paths in the transferred application package include:

App.tsx	Main route table and application layout
pages/	Public pages, authentication, dashboards, and admin screens
components/	Shared layout, controls, and homepage sections
convex/schema.ts	Data model and indexes
convex/users.ts	User sync, roles, provider profiles, and account state
convex/jobs.ts	Booking requests and service progress
convex/orders.ts	Payment order records and status
convex/reviews.ts	Completed-service reviews
convex/notifications.ts	User notifications
convex/files.ts	Upload authorization and file handling
convex/stripe.ts	Subscriptions, payments, Connect, and billing actions
convex/http.ts	Signed Stripe webhook route
convex/admin.ts	Protected administrator queries and mutations
convex/adminActions.ts	Administrator actions
convex/lib/	Shared authentication and admin authorization helpers
public/	Static public assets
.env.example	Public browser configuration placeholders
LOCAL_SETUP.md	Local service configuration notes
docs/PRODUCTION-CHECKLIST.md	Deployment and release checklist

6. Data model

users

Stores the Clerk identity, email, name, selected role, onboarding state, avatar, Stripe customer and Connect references, subscription state, and provider-specific profile fields. Provider fields include business name, phone, service areas, years in business, logo, services, fixed service prices, certifications, and verification status.

jobs

Stores a direct homeowner request linked to one provider. Important fields include homeowner ID, provider ID, title, description, pest type, location, saved amount, preferred date, photos, timestamps, and status. Supported states include requested, confirmed, declined, cancelled, scheduled, in progress, and completed.

orders

Stores the payment record linked to a job. It includes customer and provider identities, amount, Stripe payment reference, creation time, and pending, paid, or refunded state.

reviews

Stores one completed-service review with the job, provider, customer display name, rating, comment, and creation time.

notifications

Stores messages for a user, optional job linkage, read state, and creation time.

platformSettings

Stores the public platform name, logo, provider subscription amount, and the related Stripe price reference.

adminCredentials and adminSessions

Store the salted admin password hash and short-lived administration sessions. Credentials must be rotated during handover. Do not send them in public documents or commit them to source control.

contactMessages

Stores public contact submissions and their new, read, or closed status.

7. Required buyer-owned services

Create buyer-owned accounts before production launch:

- Clerk application for homeowner and provider identity.
- Convex deployment for application data, server functions, and storage.
- Stripe account for subscriptions, service payments, Connect, and webhooks.
- HTTPS web hosting for the static frontend.
- Domain registrar and DNS access for KillerBug.com.
- Email delivery service if transactional email is added or enabled.

Do not reuse seller secrets. Rotate every transferred credential and remove access that is no longer required.

8. Configuration

Create `.env.local` from `.env.example` for local work. Only public browser values belong in this file:

```
VITE_CLERK_PUBLISHABLE_KEY=YOUR_CLERK_PUBLISHABLE_KEY
VITE_CONVEX_URL=https://YOUR_DEPLOYMENT.convex.cloud
VITE_STRIPE_PUBLISHABLE_KEY=YOUR_STRIPE_PUBLISHABLE_KEY
```

Set server-only values in the buyer-owned Convex deployment:

```
CLERK_JWT_ISSUER_DOMAIN=https://YOUR_CLERK_ISSUER
STRIPE_SECRET_KEY=YOUR_SERVER_ONLY_STRIPE_SECRET
STRIPE_WEBHOOK_SECRET=YOUR_SERVER_ONLY_WEBHOOK_SECRET
HOST_URL=https://YOUR_SITE_URL
SITE_ID=YOUR_SITE_IDENTIFIER
```

Never expose server-only values through a VITE_ variable. Never place real values in documentation, screenshots, tickets, or tracked files.

9. Local setup

Use the package manager declared by the application package.

```
pnpm install --frozen-lockfile
pnpm exec convex dev
pnpm run seed
pnpm run dev
```

Run the install command only when dependencies are missing or the lockfile has changed. The seed command is optional and adds demonstration data only when appropriate. Never run demonstration seed work against a live dataset without reviewing its behavior first.

Useful checks are:

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

10. Authentication and authorization setup

1. Create the buyer-owned Clerk application.
2. Configure the production domain and allowed redirect URLs.
3. Configure the Clerk JWT template or issuer expected by Convex.
4. Add the publishable key to the frontend environment.
5. Add the issuer domain to Convex server settings.
6. Test sign-up, sign-in, sign-out, verification, recovery, and first-time role selection.
7. Confirm a homeowner cannot perform provider actions.
8. Confirm one provider cannot edit another provider's profile, services, jobs, or payout state.

Role selection is stored after onboarding. The dashboard chooses the homeowner or provider view from the stored role.

11. Payments, subscriptions, and payouts

Provider subscriptions

The owner sets the monthly provider amount through the protected billing action. Stripe creates the matching recurring price, and the application stores the price reference with platform settings. Subscription creation uses the signed-in provider and buyer-owned Stripe account.

Customer service payments

The server reads the selected job and provider records, checks that the provider has a Connect account, derives the amount from saved data, creates a Stripe PaymentIntent with an idempotency key, and creates a pending order. Payment confirmation retrieves the PaymentIntent from Stripe before marking the order paid.

Stripe Connect

Providers use a generated onboarding link to create or complete their connected payout account. Production use depends on Stripe approval, the buyer's platform settings, supported countries, and required identity checks.

Signed webhooks

Configure Stripe to send required events to:

```
https://YOUR_CONVEX_HTTP_HOST/stripe/webhook
```

Set the matching signing secret in Convex. The webhook handles subscription payment and deletion events. Test signatures and retry behavior before launch.

12. File storage

Issue photos, provider logos, and certification documents use Convex Storage. Keep authorization checks around upload URL creation and file association. Review file size, accepted file types, retention, and privacy rules before accepting public uploads.

Do not use public demonstration images as proof of real providers or completed work. Remove sample data before commercial launch.

13. Deployment sequence

1. Create the buyer-owned Clerk, Convex, Stripe, hosting, registrar, and DNS accounts.
2. Add production environment values to the correct service. Keep all secrets server-side.
3. Configure the Clerk production domain and Convex identity integration.
4. Configure Stripe products, Connect settings, webhook endpoint, and signing secret.
5. Deploy the Convex backend with type checking enabled.
6. Build the static frontend with the production Convex URL and public keys.
7. Upload the generated `dist/` contents to the HTTPS document root.
8. Include the Apache fallback configuration so direct browser routes return the single-page application.
9. Point KillerBug.com DNS to the selected host and enable HTTPS.
10. Run the full functional verification checklist before accepting real users or payments.

Shared hosting is sufficient for the static frontend when it supports HTTPS and the Apache fallback. Convex continues to host the backend functions and data.

14. Functional verification checklist

- Public pages load on desktop and mobile.
- Direct navigation works for public, auth, provider profile, dashboard, and admin routes.
- Homeowner registration and role selection succeed.
- Provider registration and role selection succeed.
- A provider can update profile, areas, services, prices, logo, and proof documents.
- The owner can review and approve or reject a provider.
- A homeowner can choose one provider and submit a request with an optional photo.
- The selected provider can accept, decline, schedule, and update the request.
- The payment amount matches the provider's saved service price.
- Stripe test payment creates one order and reaches paid state only after server confirmation.
- Connect onboarding returns to the expected production domain.
- Subscription creation, payment, cancellation, and webhook updates work in test mode.
- A completed job can receive a review, and an incomplete job cannot.
- Contact messages appear in the administration area.
- Admin sessions expire and the password can be rotated.
- Private actions reject signed-out users and users with the wrong role.
- No demonstration content is presented as a real provider, review, or business result.

15. Security and credential rotation

During handover:

1. Change the admin password through the protected administration screen.
2. Rotate Clerk, Convex, Stripe, hosting, DNS, and webhook credentials.
3. Remove seller users, access tokens, deployment access, and authorized devices that are no longer needed.
4. Keep production and test credentials separate.
5. Limit service permissions to the minimum needed.
6. Confirm no secret exists in the browser bundle, source history, logs, screenshots, or public documents.
7. Review provider document access and personal-data retention.
8. Set a written process for refunds, chargebacks, provider disputes, and security incidents.

16. Backup, monitoring, and maintenance

- Schedule regular Convex data exports and test restoration procedures.
- Keep a separate record of DNS, service ownership, and recovery contacts.
- Monitor Convex logs, failed actions, Stripe webhook delivery, payment errors, and authentication failures.
- Review dependencies and security notices on a regular schedule.
- Run targeted type checks after changes and one production build before release.
- Test the full booking and payment path after changes to auth, jobs, orders, payments, or webhooks.
- Review legal pages, provider requirements, prices, refund rules, and privacy practices whenever the launch area changes.

17. Product and support boundaries

The application package does not prove production deployment, live revenue, traffic, active providers, paying customers, payment approval, or regulatory compliance. The buyer must configure buyer-owned services, replace sample content, test every production workflow, recruit providers, market the business, and complete legal review.

Codeblix provides 30 days of hands-on support for source and domain handover, buyer-owned account setup, deployment help, current-feature questions, and small corrections. New features, major design changes, business operations, provider recruitment, marketing, and legal work are outside that support boundary.