
DriverBud Technical Handover

Buyer and technical operator guide

This guide explains the transferred application package, the buyer-owned services needed to operate DriverBud, the deployment sequence, and the checks to complete before public launch.

1. Product and system overview

DriverBud is a single-page web application for a UK driving instructor marketplace and lesson record system. It has three main areas:

- A public website and instructor directory.
- Authenticated learner and instructor workspaces.
- A separate owner operations console with its own expiring session and audit trail.

The public frontend is compiled to static files. Authentication is handled by Clerk. Application data, realtime functions, file storage, scheduled work, and HTTP endpoints are handled by Convex. Stripe is used for instructor subscriptions and featured-placement purchases.

Lesson payments are outside the platform. There is no Stripe Connect flow, lesson checkout, instructor payout, or payment splitting.

2. Application areas and roles

Public area

The public area includes the home page, instructor directory, instructor profiles, location pages, lesson-type pages, guides, pricing, instructor acquisition, about, contact, privacy, terms, and safety pages.

Learner workspace

Learners can manage onboarding details, favourites, Learning Briefs, private enquiries, messages, mutually accepted lesson proposals, active instructor relationships, lesson history, shared notes, skill progress, actions, milestones, reflections, and eligible reviews.

Instructor workspace

Instructors can manage approval information, profiles, vehicles, service areas, lesson types, availability, leads, proposals, messages, learners, lessons, skills, practice actions, analytics, subscriptions, and featured placement.

Owner console

The owner can manage users, instructor approval, verification files, profiles, marketplace data, learning categories, lesson oversight, plans, Stripe records, promotions, public content, reviews, reports, privacy requests, contact requests, expiry notices, platform settings, and audit records.

3. Architecture

Browser application

The frontend uses React 19, TypeScript, Vite, React Router, Tailwind CSS, GSAP, Lenis, and Lucide React. It is built as static HTML, CSS, JavaScript, fonts, and images.

Browser code receives only public values:

- The Convex public deployment URL.
- The Clerk publishable key.
- The Stripe publishable key.
- The public site URL.

Never place a secret key in a browser variable or any variable whose name starts with VITE_.

Identity

Clerk provides learner and instructor identity. Convex validates the Clerk identity and enforces role and ownership checks in server functions.

The owner console does not use the normal Clerk session. It uses a separate username and password configured only in the Convex server environment. Successful sign-in creates an expiring server-side session. Privileged actions create audit records.

Backend and data

Convex provides the document database, server queries and mutations, realtime updates, file references, scheduled tasks, and HTTP actions. Public queries return only suitable public data. Private learner details, messages, proposals, verification files, and instructor-private notes are scoped by server-side checks.

Billing

Stripe Checkout and Billing handle instructor plans and featured placement. The server creates or reuses Stripe customers and products, starts checkout sessions, verifies webhook signatures, records events idempotently, and updates plan entitlements.

The webhook route is:

```
https://YOUR_CONVEX_SITE_URL/stripe/webhook
```

Use the HTTP actions domain provided by the buyer's Convex deployment. Do not use the frontend domain for this webhook unless it is explicitly routed to the Convex action.

Static hosting

The production build is stored in `dist/`. It can be published to ordinary static or shared hosting. The included `.htaccess` file sends unknown browser routes to `index.html` and adds basic security headers. The backend still runs in the buyer's Convex account.

4. Project structure

The main relative paths are:

src/	
App.tsx	Main route and application shell
publicPages.tsx	Public website and discovery pages
dashboardPages.tsx	Learner and instructor workspaces
adminPage.tsx	Separate owner operations console
components.tsx	Shared interface components
product.ts	Public brand and product configuration
lib.ts	Shared browser helpers
index.css	Global styles and design system
convex/	
schema.ts	Data model and indexes
auth.config.ts	Clerk identity provider configuration
http.ts	Stripe webhook and health routes
billing.ts	Checkout and billing operations
adminAuth.ts	Owner sign-in and session management
admin.ts	Owner data and privileged actions
users.ts	Account and role operations
instructors.ts	Instructor profiles and approval flows
learners.ts	Learner workflows
requests.ts	Learning Briefs and lesson requests
messages.ts	Scoped conversations and messages
operations.ts	Reports, privacy, contact, and audit work
content.ts	Public content management
seed.ts	Manual, idempotent sample-data seed
public/	
.htaccess	Shared-hosting route fallback and headers
fonts/	Included font files and licences
logo and favicon files	Public brand assets
.env.example	Safe configuration template
package.json	Scripts, dependencies, and pnpm version
vite.config.ts	Frontend build configuration

5. Data model

The main data groups are listed below. Exact fields and indexes are defined in `convex/schema.ts`.

Identity and instructor records

- User accounts hold the Clerk identity, role, profile details, and status.
- Instructor profiles hold owner identity, public address, approval status, services, areas, policies, media references, and marketplace metrics.
- Team membership records connect staff members to a driving-school account and role.
- Vehicle records hold transmission, adaptations, public notes, and status.
- Private verification records hold a protected file reference, type, expiry, and review status.
- Availability records hold weekly windows and date-specific changes.

Discovery, requests, and conversations

- Learning Brief records hold the learner goal, preferences, schedule, pickup area, budget, and status.
- Lead records connect one instructor to a Learning Brief and hold match details and pipeline status.
- Lesson proposal records hold the proposed schedule, location, terms, expiry, and both acceptance states.

-
- Conversation records are scoped to a lead or learner relationship.
 - Message records hold the sender, body, protected attachment references, reports, and timestamps.

Lesson and progress records

- Learner relationship records connect an instructor and learner and hold goals, shared notes, private notes, and status.
- Lesson records hold schedule, status, focus, observations, shared notes, private notes, and learner reflection.
- Skill records hold evidence and ratings linked to a lesson or learner relationship.
- Practice-action records hold the assigned task, owner, due date, and status.
- Learning configuration holds owner-managed skill groups, labels, exercises, and milestone rules.

Billing and marketplace records

- Plan records hold a plan code, price, interval, features, limits, Stripe mapping, and active state.
- Subscription records hold the customer, Stripe identifiers, plan, status, billing period, and entitlement state.
- Promotion records hold the featured-placement period, amount, and Stripe state.
- Webhook records provide idempotent receipt and processing information.
- Billing history records subscription, payment, refund, and promotion activity for review.
- Review records hold eligible learner feedback and moderation status.

Content and operations

- Content records hold location, lesson, guide, FAQ, testimonial, safety, navigation, and footer content.
- Contact records hold public messages and handling status.
- Report records hold user or content concerns and their resolution state.
- Privacy records hold export or deletion requests and deadlines.
- Notification records support account and workflow updates.
- Audit records hold privileged and important workflow actions.
- Settings records hold owner-controlled platform options.

6. Buyer-owned services

Create new buyer-owned accounts or projects for:

- Clerk authentication.
- Convex backend, data, files, schedules, and HTTP actions.
- Stripe payments and billing.
- Static or shared web hosting.
- Domain and DNS management.

Do not depend on seller-owned credentials after the handover. Rotate every transferred value before public operation.

7. Configuration

Browser configuration

Copy `.env.example` to `.env.local` for local work. Use public values only:

```
VITE_CONVEX_URL=https://YOUR_DEPLOYMENT.convex.cloud
VITE_CLERK_PUBLISHABLE_KEY=pk_test_YOUR_PUBLIC_KEY
VITE_STRIPE_PUBLISHABLE_KEY=pk_test_YOUR_PUBLIC_KEY
VITE_SITE_URL=https://driverbud.com
```

The publishable keys are intended for browser use. They are still buyer-specific and should be replaced during handover.

Convex server configuration

Set these values in the buyer's Convex deployment environment:

```
CLERK_JWT_ISSUER_DOMAIN=https://YOUR_CLERK_ISSUER
STRIPE_SECRET_KEY=sk_test_YOUR_SERVER_SECRET
STRIPE_WEBHOOK_SECRET=whsec_YOUR_WEBHOOK_SECRET
SITE_URL=https://driverbud.com
APP_ID=YOUR_APPLICATION_ID
APP_NAME=DriverBud
ADMIN_INITIAL_USERNAME=YOUR_OWNER_USERNAME
ADMIN_INITIAL_PASSWORD=YOUR_LONG_RANDOM_PASSWORD
```

Optional owner allowlist and webhook-forwarding settings should be added only if the buyer needs those features. Keep every server secret out of `.env.local`, the browser build, public logs, screenshots, and support messages.

8. Local setup

Use Node.js 22 and the pnpm version declared in `package.json`.

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

Open the local address shown by Vite. Keep the Convex development command running only during active development.

The manual seed adds presentation and workflow data. Run it only against the intended buyer deployment and only after reviewing its purpose:

```
pnpm run seed
```

The seed is idempotent and adds missing sample records, but sample users and content still need review before a public launch.

9. Clerk setup

- 1 Create a buyer-owned Clerk application.
- 2 Copy its publishable key into the browser configuration.
- 3 Configure the Convex integration and JWT issuer domain in the Convex environment.
- 4 Enable the sign-in methods the buyer intends to support.
- 5 Set production domains and redirect URLs for DriverBud.com.

-
- 6 Test new learner and instructor registration.
 - 7 Confirm one user cannot read or change another user's private records.

The owner console uses separate server-side credentials and should not be presented as a Clerk account.

10. Convex setup

- 1 Create a buyer-owned Convex project.
- 2 Authenticate the Convex command-line tool with the buyer account.
- 3 Set all server environment values in the development deployment.
- 4 Run the development command and allow Convex to deploy the schema and functions.
- 5 Test the health route at `https://YOUR_CONVEX_SITE_URL/health`.
- 6 Review file limits, scheduled work, usage, and retention requirements.
- 7 Repeat the configuration in the production deployment with production values.

Do not copy production secrets into development values or browser files.

11. Stripe setup

- 1 Create or use a buyer-owned Stripe account.
- 2 Begin in Stripe test mode.
- 3 Set the Stripe publishable key in the frontend build environment.
- 4 Set the Stripe secret key in the Convex server environment.
- 5 Add the Convex HTTP actions webhook URL ending in `/stripe/webhook`.
- 6 Subscribe the webhook to the checkout, subscription, invoice, and refund events used by the application.
- 7 Copy the webhook signing secret to the Convex server environment.
- 8 Open the owner billing area and review plan names, prices, intervals, limits, and active state.
- 9 Complete one instructor subscription checkout and one featured-placement checkout in test mode.
- 10 Confirm webhook records are processed once and the expected entitlement or promotion becomes active.
- 11 Switch to live keys only after the full test checklist passes.

Stripe handles instructor subscriptions and promotions only. Do not advertise lesson checkout or instructor payouts unless those features are separately developed and tested.

12. Owner console setup

- 1 Set a unique initial owner username and a strong password in the Convex production environment.
- 2 Sign in through the separate owner route supplied with the application.
- 3 Change the credentials from inside the owner console immediately.
- 4 Confirm existing sessions are revoked after a credential change.
- 5 Review the audit log after approvals, plan changes, privacy actions, and other privileged work.

-
- 6 Restrict owner access to trusted operators and do not share one account across a large team.

Never place owner credentials in frontend variables, public documentation, or source files.

13. Build and deployment

Recommended automated deployment

The included workflow is designed for production updates. It installs locked dependencies with pnpm, deploys Convex, builds the frontend with the deployed public URL, and can publish `dist/` to shared hosting over FTPS.

Create encrypted deployment secrets in the buyer's source-control or hosting system for:

- Convex deploy access.
- Clerk and Stripe publishable keys.
- Public site URL.
- FTPS host, username, password, and target directory when FTPS publishing is used.

Keep source-control permissions read-only except where the workflow needs deployment access.

Manual frontend deployment

- 1 Configure the production browser values.
- 2 Deploy the production Convex functions.
- 3 Run one production build:

```
pnpm run build
```

- 1 Upload the contents of `dist/` to the web root for DriverBud.com.
- 2 Confirm `.htaccess` is included when using Apache shared hosting.
- 3 Enable HTTPS and point DNS to the chosen host.
- 4 Open a nested route directly to confirm the rewrite fallback works.

Shared hosting is suitable for the static frontend. It does not replace the Convex backend, Clerk authentication, or Stripe billing services.

14. Security and credential rotation

- Replace every seller-owned key and deployment token with a buyer-owned value.
- Keep Stripe, webhook, owner, and deployment secrets in server-side secret stores.
- Keep private verification files restricted to authorized workflows.
- Review role, ownership, approval, and admin checks after any backend change.
- Keep webhook signature verification enabled.
- Preserve idempotency checks so repeated webhooks do not grant duplicate entitlements.
- Use strong owner credentials and rotate them after personnel changes.
- Keep audit records for privileged actions and review unusual activity.

-
- Review rate limits for inquiries, messages, reviews, sign-in attempts, and uploads.
 - Validate file type, size, access, and expiry metadata.
 - Publish clear privacy, retention, deletion, safety, and marketplace terms before launch.
 - Do not store government identity or driving evidence unless the buyer has a lawful need and suitable controls.

15. Functional verification checklist

Public and account checks

- Home page, instructor directory, profile pages, content pages, and nested routes load over HTTPS.
- Learner and instructor registration create the correct role.
- Unapproved instructors do not appear publicly.
- Private pages reject signed-out users and users with the wrong role.

Learner checks

- A learner can save a profile and submit a Learning Brief.
- Separate instructor invitations create separate private conversations.
- The learner can accept a proposal and view only permitted lesson information.
- Reflections, confidence, actions, progress, and eligible reviews behave correctly.

Instructor checks

- An instructor can submit a profile and private verification information.
- Approval is required before public publication.
- Availability, leads, messages, proposals, relationships, lessons, skills, and actions can be updated only by authorized users.
- Analytics and plan limits reflect current records.

Billing checks

- Test subscription checkout returns to the correct site.
- A signed webhook changes the matching subscription entitlement.
- Replaying the same webhook does not duplicate the result.
- A test featured-placement purchase activates the expected period.
- Failed or unsigned webhook requests are rejected and recorded appropriately.

Owner checks

- Owner sign-in uses an expiring session.
- Credential changes revoke existing sessions.
- Instructor approval, plan edits, moderation, privacy actions, and settings changes create audit records.
- Private instructor notes and verification files are not exposed through public queries.

Hosting checks

- `pnpm run build` completes without a type error.
- Refreshing a nested route returns the application instead of a 404 page.
- Browser developer tools show no secret values in built files or network responses.
- Security headers and HTTPS are active.

16. Backup, monitoring, and maintenance

- Configure regular Convex backups or exports under the buyer's account and test a restore procedure.
- Keep a separate record of Clerk and Stripe configuration that does not include plain-text secrets.
- Monitor Convex function errors, HTTP action failures, scheduled jobs, file usage, and database growth.
- Monitor Stripe webhook delivery, failed invoices, refunds, disputes, and subscription status.
- Review expiring verification files and open reports regularly.
- Review privacy and deletion requests before their due dates.
- Update dependencies in planned batches, review release notes, run targeted checks, then complete one production build before deployment.
- Test role boundaries, billing, and private data access after dependency or schema changes.
- Keep public legal and safety content aligned with the operating region.

17. Product boundaries

DriverBud includes instructor discovery, Learning Briefs, private messages, mutual lesson confirmation, lesson records, skill evidence, planning aids, instructor subscriptions, featured placement, content management, and owner operations.

It does not include lesson payments, deposits, payouts, Stripe Connect, maps, GPS routes, telematics, video analysis, government test booking, licence verification APIs, insurance validation, email or SMS delivery services, native mobile apps, or generative AI.

The buyer is responsible for service accounts, hosting, data protection, instructor checks, legal review, moderation, marketing, customer support, and ongoing operating costs.

18. Handover and support

Codeblix provides up to 30 days of reasonable technical handover support for transfer questions and initial setup guidance. Support does not include unlimited custom development, ongoing operations, third-party fees, legal advice, marketing, or guaranteed commercial results.