

ThatQuis Technical Handover

This document explains the transferred application package for the buyer or technical operator. It covers the system design, project structure, data model, buyer-owned service configuration, deployment, security, testing, backups, and maintenance.

1. Product and system overview

ThatQuis is a subscription SaaS for building and publishing quiz funnels. Creators use a private workspace to build versioned quizzes, collect consented leads, and review analytics. Respondents complete public quizzes without an account. A separate platform administration area manages creators, plans, templates, website content, moderation, privacy work, usage, and billing operations.

The frontend is a React 19 single-page application written in TypeScript and built with Vite. React Router handles public pages, creator pages, public quiz links, result links, and administration routes. The compiled frontend is static and can be hosted on ordinary shared hosting. Vercel is not required.

Convex provides the managed backend, database, realtime queries, server functions, storage, scheduled work, and HTTP endpoints. Clerk provides creator authentication. Stripe provides subscription checkout, the customer billing portal, and payment webhooks.

2. Main application areas and roles

Public marketing area

The public site contains the homepage, pricing, features, templates, examples, company pages, contact page, privacy policy, terms, and acceptable-use content. Much of this content can be managed from the platform administration area.

Creator workspace

Authenticated creators can complete onboarding, manage a workspace, create quizzes, edit branching and scoring, configure outcomes and lead consent, publish quiz links, inspect responses, review analytics, export allowed data, manage privacy work, maintain branding, invite permitted team members, and manage billing.

Public quiz runner

Respondents open a published quiz through a public slug. The backend exposes only respondent-safe published content. A short-lived session stores progress, answers, and source information. Server logic calculates the outcome and issues a scoped result token.

Platform administration

The administration area uses its own credential and expiring session, separate from creator authentication. It manages creators, plans, subscriptions, templates, public content, reports, privacy requests, contacts, usage, audit records, operational settings, and maintenance mode. Change the transferred bootstrap credential before launch.

3. Technology stack

- React 19 and React DOM for the user interface.

- TypeScript for typed frontend and backend code.
- Vite for local development and static production builds.
- React Router for browser routing.
- Convex for application data, server functions, realtime updates, storage, scheduled jobs, and HTTP actions.
- Clerk for creator authentication and identity tokens.
- Stripe for subscription checkout, customer billing management, and signed webhook events.
- Tailwind tooling and application CSS for styling.
- Lucide React for interface icons.

4. Project structure

- `src/main.tsx` creates the application providers and starts the browser application.
- `src/App.tsx` defines the primary route map and authentication boundaries.
- `src/home/` contains the homepage and its supporting sections.
- `src/pages/` contains authentication, onboarding, public templates, and marketing pages.
- `src/workspace/` contains the creator dashboard, quizzes, responses, analytics, billing, settings, and privacy screens.
- `src/builder/` contains the quiz editor, panels, question editor, image upload, and persisted builder state.
- `src/runner/` contains the public quiz flow, lead form, result handling, and scoped result token logic.
- `src/admin/` contains the platform administration interface.
- `src/domain/` contains shared product types, defaults, and rules.
- `src/ui/` contains reusable interface elements.
- `convex/schema.ts` defines the managed data model and indexes.
- `convex/auth.config.ts` configures identity-token verification.
- `convex/users.ts`, `convex/workspaces.ts`, and `convex/quizzes.ts` implement creator and workspace operations.
- `convex/templates.ts`, `convex/seed.ts`, and `convex/seedData.ts` implement the template library and guarded demonstration seed.
- `convex/publicQuiz.ts`, `convex/publicForms.ts`, and `convex/responses.ts` implement public quiz sessions, submissions, consented leads, and result access.
- `convex/analytics.ts` provides creator and platform reporting queries.
- `convex/plans.ts`, `convex/plansInternal.ts`, `convex/billing.ts`, and `convex/stripeEvents.ts` implement plans, entitlements, checkout, portal sessions, and webhook processing.
- `convex/privacy.ts`, `convex/maintenance.ts`, and `convex/crons.ts` implement privacy, retention, maintenance, and scheduled work.

Administration and HTTP modules

- `convex/admin.ts` and `convex/cms.ts` implement administration and editable public content.
- `convex/http.ts` exposes HTTP endpoints, including Stripe event intake and routed webhook behavior.
- `public/` contains brand assets, fonts, screenshots, and shared-hosting route fallback configuration.

5. Data model

Identity and workspaces

- `users` stores the Clerk subject, profile details, account state, and onboarding state.
- `workspaces` stores the owner, workspace name, plan, branding, and monthly usage state.
- `memberships` stores user-to-workspace roles and membership status.
- `teamInvitations` stores pending and completed invitations.
- `brandKits` stores reusable colors, typography, button choices, and uploaded brand assets.

Plans and billing

- `plans` stores public plan labels, prices, limits, entitlements, visibility, and Stripe mappings.
- `subscriptions` stores Stripe customer and subscription references, subscription periods, and status.
- `usageLedger` stores counted response usage for enforcement and reporting.
- `stripeEvents` or `webhook-event` records support idempotent event processing and operational review.

Prices should be stored in integer minor units where required. Stripe identifiers are references, not secrets. Secret keys and webhook signing secrets belong only in the managed server environment.

Quiz authoring

- `quizzes` stores workspace ownership, slug, title, goal, status, settings, and the current version.
- `quizVersions` stores immutable published snapshots so a live quiz does not change while responses are being recorded.
- `questions` stores question type, content, options, order, branching, scoring, and outcome tags.
- `outcomes` stores ranges or rules plus the result title, summary, recommendations, call to action, image, and sharing text.

Respondent data

- `responseSessions` stores the quiz version, anonymous session token hash, progress, source, status, and timing.
- `answers` stores normalized answers and timing for each response session.
- `leads` stores only the consented fields, related quiz and workspace, tags, and notes.
- `resultViews` stores scoped result-token hashes, outcomes, and expiry.
- `analyticsEvents` stores funnel and interaction events used for aggregate reporting.

Operations and content

- `templates` stores editable quiz templates and their display metadata.

- `cmsEntries` stores public site content and brand settings.
- `reports` stores reported quiz content and moderation outcomes.
- `privacyRequests` stores respondent and workspace privacy work.
- `contactRequests`, `notifications`, and `auditLogs` support operations and accountability.
- `adminCredentials` and `adminSessions` support the separate administration gate.
- `seedRuns` prevents the same demonstration seed from running more than once.

6. Authentication and authorization

Creator sign-in uses Clerk. Convex verifies the identity token and server functions resolve the matching user. Private reads and writes must check workspace membership and the required role. Do not trust a workspace identifier supplied by the browser without a server-side membership check.

Public quiz queries must return only the active published version and fields needed by the respondent. Draft content, private scoring details, other respondents, and creator data must remain unavailable.

The platform administration area uses a separate username and password flow. A successful login returns an expiring session token used by administration queries and mutations. The session is kept in browser session storage and expires after 12 hours. Change the initial credential during handover and invalidate older sessions after the change.

7. Buyer-owned services and configuration

Create buyer-owned accounts for Convex, Clerk, Stripe, the domain registrar, and shared hosting. Use least-privilege access and enable multifactor authentication where available.

Create an ignored `.env.local` file for public browser 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
```

Set server-only values in the Convex deployment environment:

```
CLERK_JWT_ISSUER_DOMAIN=https://YOUR_CLERK_ISSUER
STRIPE_SECRET_KEY=YOUR_STRIPE_SECRET_KEY
STRIPE_WEBHOOK_SECRET=YOUR_STRIPE_WEBHOOK_SECRET
SITE_URL=https://YOUR_SITE_URL
APP_ID=thatquis
APP_NAME=ThatQuis
CONVEX_SITE_ID=YOUR_CONVEX_SITE_ID
ADMIN_CLERK_IDS=OPTIONAL_LEGACY_ADMIN_IDS
ADMIN_EMAILS=OPTIONAL_LEGACY_ADMIN_EMAILS
```

Values beginning with `VITE_` are included in the browser bundle and must never contain private secrets. Keep payment secrets, webhook secrets, and administrative server settings in Convex only. Do not commit `.env.local` or export real values into documentation.

8. Setup and installation

Use the existing dependency directory when it is valid. If dependencies are missing, use `pnpm`:

```
pnpm install
```

Do not use npm for this package. The buyer may remove an obsolete npm lockfile after creating and validating a pnpm lockfile, but should not change lockfiles during an unrelated production incident.

For local frontend work:

```
pnpm run dev
```

For Convex development and function synchronization:

```
pnpm exec convex dev
```

Copy the example environment file to `.env.local`, add only buyer-owned values, configure the Clerk issuer in Convex, and confirm creator sign-in before testing private workflows.

9. Initial data and demonstration content

The guarded seed creates three plans, 18 templates, eight published examples, 150 demonstration responses, 70 demonstration leads, brand kits, frequently asked questions, testimonials, analytics history, and two demonstration workspaces.

Run it once from an authorized environment:

```
pnpm run seed
```

The seed uses a recorded marker and should not recreate the same data on later runs. Review the seed implementation and the target deployment before running it. Remove or clearly label demonstration personal data before handling live customer information.

10. Stripe setup

- 1 Create buyer-owned Free, Pro, and Agency definitions in the platform administration area.
- 2 Create matching recurring products and prices in Stripe for paid plans.
- 3 Map each paid plan to its Stripe price identifier.
- 4 Set `STRIPE_SECRET_KEY` only in the Convex server environment.
- 5 Configure the webhook endpoint used by the deployed application.
- 6 Set `STRIPE_WEBHOOK_SECRET` only in the Convex server environment.
- 7 Confirm that shared Stripe metadata includes the application identifier and site identifier expected by the event router.
- 8 Test checkout, successful payment, plan activation, billing portal access, cancellation, failed payment, webhook replay, and duplicate-event handling in Stripe test mode.
- 9 Switch to live credentials only after test-mode verification passes.

Stripe is for creator subscriptions paid to the platform owner. There is no Stripe Connect workflow and no payment collection from respondents for creators.

11. Deployment guide

Deploy managed backend functions

Authenticate the buyer's Convex account, select the intended deployment, set the server environment values, and deploy:

```
pnpm exec convex deploy
```

Record the resulting public Convex URL in `.env.local`. Confirm that authentication settings refer to the buyer's Clerk issuer.

Build the static frontend

Run one production build after configuration is complete:

```
pnpm run build
```

The command performs a TypeScript check and produces the static `dist/` directory.

Upload to shared hosting

Upload the contents of `dist/` to the document root for ThatQuis.com. Preserve the included Apache history fallback so direct visits and browser refreshes work for creator routes, public quiz routes, and result routes. Shared hosting is sufficient for the frontend because Convex provides the managed backend. No Vercel deployment is required.

Configure HTTPS, redirect the preferred domain form, and update `SITE_URL` plus `VITE_SITE_URL` to the final HTTPS address. Test a direct visit to a nested marketing page, a creator route, a public quiz link, and a result link.

Configure the domain

Point the domain to the selected hosting account, enable SSL, and wait for DNS propagation. Do not change live DNS until the new deployment has been tested on a temporary address or local host mapping.

12. Security and credential rotation

- Transfer or recreate every service under buyer ownership.
- Rotate the platform administrator credential immediately.
- Rotate Stripe keys and webhook signing secrets after transfer.
- Remove seller access from the registrar, hosting, Clerk, Convex, Stripe, and source-control service when handover is complete.
- Keep secrets out of browser variables, application logs, screenshots, documentation, and support messages.
- Require workspace membership and role checks for every private server operation.
- Keep public quiz responses scoped by short-lived session and result tokens.

Request and file safeguards

- Validate and sanitize creator-supplied text shown on public pages.
- Validate file type and size before storing uploaded images.
- Rate-limit public session creation, submissions, access-code attempts, contact forms, and authentication attempts.
- Preserve signature verification and idempotency for Stripe webhooks.
- Review consent text, retention duration, export rights, deletion handling, terms, privacy policy, and acceptable-use policy with qualified advisers.

13. Functional verification checklist

Public and account flows

- The homepage, pricing, features, templates, examples, and legal pages load over HTTPS.
- Creator sign-up and sign-in complete through the buyer-owned Clerk instance.
- A new creator can finish onboarding and reach the workspace.
- Direct links and browser refreshes work on nested routes.

Quiz authoring

- A creator can start from a template, duplicate a quiz, and create a blank quiz.
- All supported question types save and preview correctly.
- Scoring, branching, lead timing, consent, and outcomes can be edited.
- Validation catches loops, unreachable questions, missing outcomes, and empty branches.
- Publishing creates a unique public slug and an immutable version.
- Pause, duplicate, archive, and date or response-cap controls work.

Respondent and privacy flow

- An anonymous visitor can start, resume, complete, and submit a quiz.
- Server-side outcome calculation returns the expected result.
- A scoped result link refreshes without exposing another respondent.
- Consented lead fields appear in the creator workspace.
- A privacy request reaches the creator or administrator queue and can be resolved.

Analytics and export

- Views, starts, progress, completions, leads, outcomes, and call-to-action clicks update.
- Drop-off, answer distribution, devices, referrers, UTM data, and date filters display correctly.
- Response search, filters, notes, tags, deletion, and permitted CSV export work.
- Plan limits correctly block or allow exports, branding changes, quiz publishing, responses, and team seats.

Billing and administration

- Test checkout creates the correct subscription.
- Webhooks update entitlements once even when Stripe retries an event.
- Billing portal, cancellation, failed payment, and grace-period behavior work.
- The platform administrator can manage creators, plans, templates, CMS content, reports, privacy requests, contacts, usage, and maintenance mode.
- Audit records are created for sensitive administrative changes.

14. Backup, monitoring, and maintenance

Use the backup and export options available from the buyer's managed-service plans. Before a large migration or destructive maintenance task, export critical content and verify that the export can be read. Keep a versioned copy of the application package and deployment configuration without secrets.

Monitor failed authentication, public submission rates, backend function errors, Stripe webhook failures, subscription mismatches, response usage, privacy queues, reports, storage growth, and scheduled retention work. Review service dashboards after each deployment.

Update dependencies in a separate branch or maintenance window. Read release notes, test creator and respondent workflows, run one final production build, and deploy only after the verification checklist passes. Avoid reinstalling dependencies or rebuilding repeatedly when neither source nor lockfiles changed.

15. Product and support boundaries

The current product does not include generative AI, email or SMS sending, automated CRM integrations, automated custom-domain setup, exam proctoring, course hosting, native applications, scraping, third-party outbound webhooks, Stripe Connect, or payments from respondents to creators.

The buyer is responsible for buyer-owned service accounts, deployment, live payment activation, legal review, content review, ongoing security, monitoring, support, and marketing. Codeblix provides 30 days of reasonable technical handover support for configuration, deployment, and questions about the delivered product. New features and ongoing operations require a separate agreement.