Jon’s Trade Corner — Developer & QA Handover Guide

> **Audience:** External engineering / QA team reviewing and stress-testing the build. **Assume no prior context.** Everything you need to understand, run, and review the system is in this document. **Last updated:** May 2026.

Table of contents

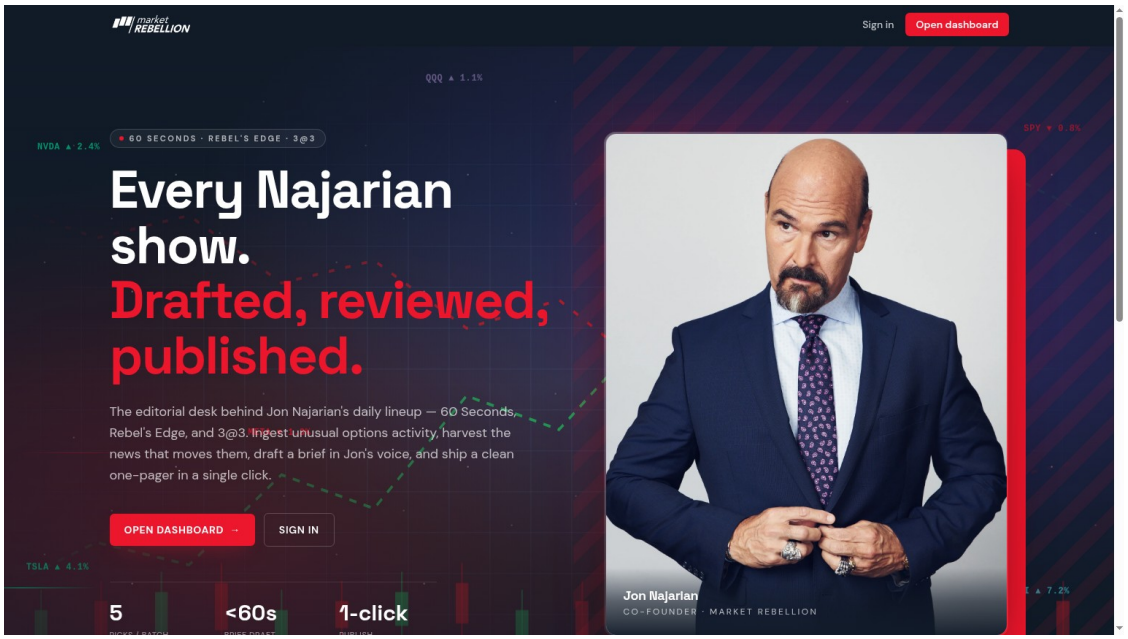
---

1. Overview

Jon’s Trade Corner is the editorial automation desk that produces Jon Najarian’s daily Market Rebellion trading briefs. Jon and the Market Rebellion analyst team use it to turn raw unusual options activity (UOA) into a polished, shareable one-pager — without losing Jon’s voice.

The problem it solves: Jon’s daily shows (3@3, Rebel’s Edge, 60 Seconds, and ad-hoc alerts) need a tight, well-researched brief per pick within minutes of the UOA signal firing. Manually researching news, writing in Jon’s voice, formatting a page, and embedding a chart used to take 20–40 minutes per ticker. This app compresses that to roughly a minute per pick: it ingests the UOA row, harvests the relevant news, drafts the brief with a Najarian-voice LLM prompt, lets a reviewer edit it, and publishes a clean public page with an embedded TradingView chart.

The primary users are Jon himself (the analyst), the Market Rebellion editorial reviewers (staff), and the public readers of the published briefs (anonymous visitors who land on `/brief/$slug`). Staff operate through `/corner`; the public consumes `/brief/$slug` and the combined `/show/$service/$date` pages.



Public landing page at /. Marketing entry point with a clear sign-in / open-dashboard call to action.

---

2. Tech stack

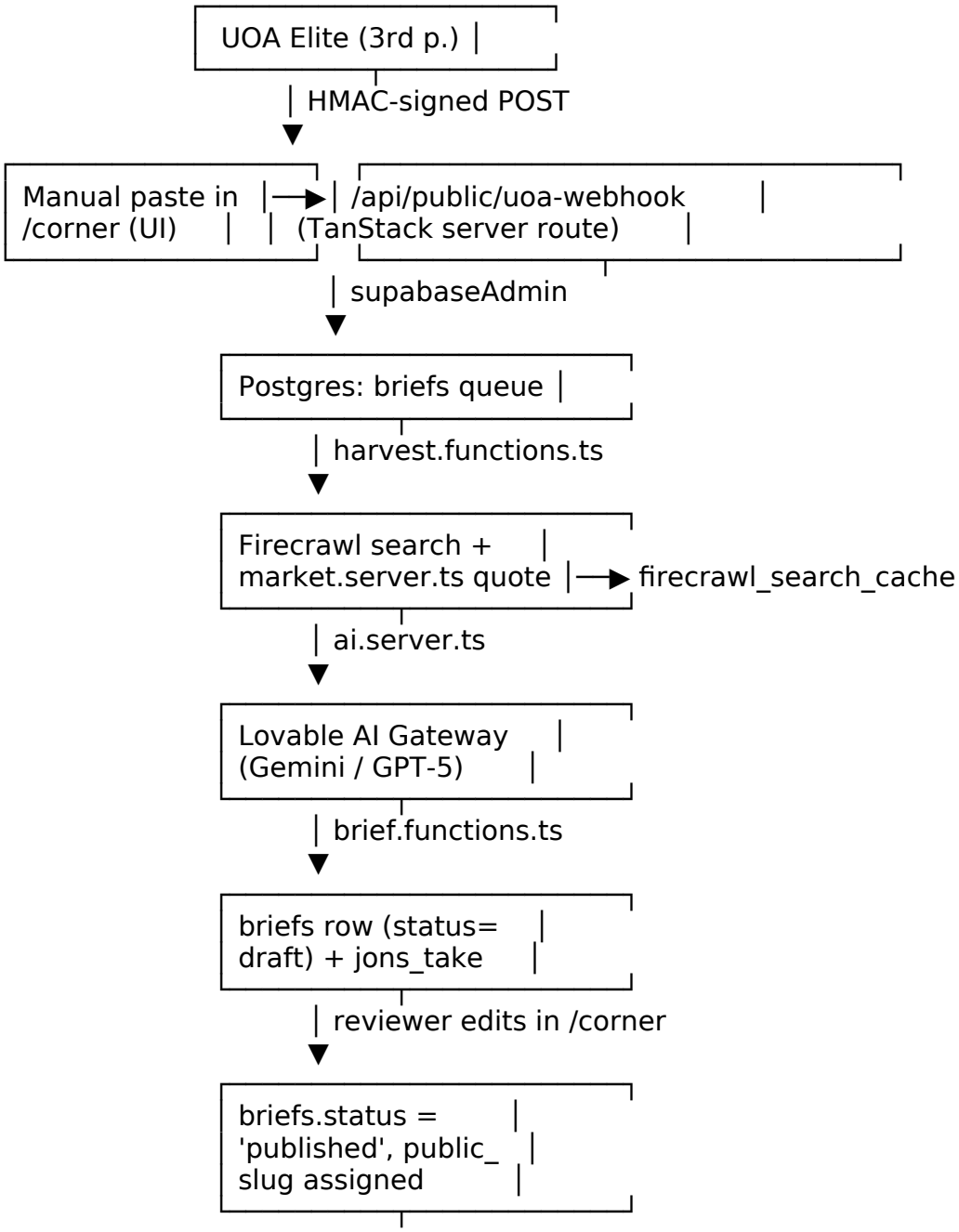
Layer	Technology	What it does in this app
Framework	**TanStack Start v1** (React 19)	Full-stack React framework with SSR + file-based routing. Replaces Next/Remix.
Build tool	**Vite 7**	Dev server and production bundler. Configured via vite.config.ts.
Runtime	**Cloudflare Workers** (nodejs_compat)	Where SSR and all server functions execute in production.
Styling	**Tailwind CSS v4** + **shadcn/ui**	Utility CSS + accessible primitive components. Tokens in src/styles.css.
State / data	**TanStack Query**	Server state, caching, suspense reads in components.
Router	**TanStack Router** (file-based)	Routes in src/routes/, auto-generated routeTree.gen.ts.

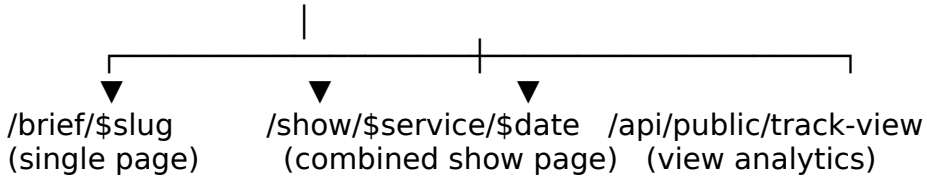
Layer	Technology	What it does in this app
Backend platform	**Lovable Cloud** (Supabase under the hood)	Postgres database, auth, file storage.
Database	**Postgres** (Supabase)	All persistent data. RLS enforced on every table.
Auth	**Supabase Auth**	Email/password + Google OAuth. Password reset emails.
AI gateway	**Lovable AI Gateway**	Single endpoint for google/gemini-2.5-flash, google/gemini-2.5-pro, openai/gpt-5-mini. No third-party LLM keys stored.
News harvest	**Firecrawl** (connector)	Search + scrape relevant news per ticker. Results cached in firecrawl_search_cache.
Charts	**TradingView** embed widget	Embedded on /brief/$slug. No API key.
Slides	**Google Slides API** (connector)	Optional slide deck export for a published brief (slides.functions.ts).
Inbound webhook	**UOA Elite**	HMAC-signed POST to /api/public/uoa-webhook to seed the brief queue.
Outbound email	**Supabase Auth emails** (default templates)	Verification, password reset. Custom sender domain not yet configured — see §7.
Server transport	**TanStack server functions** (createServerFn) + **server routes** (createFileRoute server.handlers)	All app-internal RPC and external HTTP endpoints. **No Supabase Edge Functions are used for app logic.**
Validation	**Zod**	Input validation on every server function and webhook.
Lint	ESLint + Prettier	Standard config in repo

Layer	Technology	What it does in this app
		root.
Package manager	**Bun** (bun.lockb, bunfig.toml)	Install and run dev.

3. Architecture

High-level data flow





Step-by-step walkthrough

1. **Ingest.** A UOA pick arrives either through the inbound webhook (/api/public/uoa-webhook) or by a reviewer pasting a UOA Elite row into the guided flow at /corner. The webhook handler verifies the HMAC signature with UOA_WEBHOOK_SECRET and uses supabaseAdmin to insert a new row into briefs with status='queued'.
2. **Harvest.** harvest.functions.ts calls firecrawl.server.ts to pull the top news items for the ticker. Results are cached in firecrawl_search_cache so the same ticker queried twice in the same window doesn't re-bill Firecrawl. In parallel, market.server.ts fetches the live quote / sector / company name.
3. **Draft.** brief.functions.ts builds the prompt (system prompt + few-shots from chameleon.server.ts), calls ai.server.ts which routes to the Lovable AI Gateway (google/gemini-2.5-flash by default), and writes the parsed JSON into the briefs row — thesis, bull_case, bear_case, technical_setup, risk_parameters, jons_take.
4. **Review.** A staff user opens /corner, sees the draft, and can edit inline, regenerate any section, or escalate to the advanced editor.
5. **Publish.** Hitting publish flips briefs.status to 'published' and assigns public_slug (and optionally combined_slug for grouped show pages). Anonymous public reads of /brief/$slug are allowed by RLS only when status='published'.
6. **Serve.** Public visitors land on /brief/$slug. The page server-fetches the brief, lazy-loads the TradingView chart, and the browser fires /api/public/track-view to record a view. Ask Jon is available as a streaming SSE conversation via /api/jon-ask-stream.
7. **Combined show page.** When multiple briefs share a combined_slug, /show/$service/$date (or /combined) renders them as a single show wrap.

Trust boundaries

- **Browser** → uses @/integrations/supabase/client with the publishable key. RLS applies.
- **Authenticated server functions** → use requireSupabaseAuth middleware. Run as the signed-in user. RLS applies.
- **Admin server code** (webhooks, scheduled jobs) → uses @/integrations/supabase/client.server with the service-role key. **Bypasses RLS.** Only used after signature/auth verification.

---

4. Connections and integrations

Every external connection and the secret it needs. Reviewers should confirm each one resolves cleanly and that secrets are never reachable from browser bundles.

Connection	Direction	Endpoint / package	Secret(s) used	How to verify
Lovable Cloud (Supabase) — DB + Auth + Storage	Outbound	Auto-wired	SUPABASE_URL, SUPABASE_PUBLISHABLE_KEY (server), SUPABASE_SERVICE_ROLE_KEY (admin), VITE_SUPABASE_* (browser)	Sign in at /login. Confirm select 1 works via Lovable Cloud → Database.
Lovable AI Gateway	Outbound	https://ai.gateway.lovable.dev	LOVABLE_API_KEY	Trigger a brief regenerate from /corner; check Worker logs for a 200 from the gateway.
Firecrawl	Outbound	firecrawl.dev REST API	FIRECRAWL_API_KEY (managed via Connector)	Run a harvest from /corner; new rows should land in firecrawl_search_cache.
Google Slides API	Outbound	Google APIs	GOOGLE_SLIDES_API_KEY (managed via Connector)	Trigger slides export from advanced editor; check returned URL opens.
TradingView	Outbound (client-side)	s3.tradingview.com/tv.js	None	Open /brief/$slug; chart must render. No API key.
UOA Elite	**Inbound**	POST	UOA_WEBHOO	curl a signed

Connection	Direction	Endpoint / package	Secret(s) used	How to verify
webhook		/api/public/uoa-webhook	K_SECRET (HMAC SHA-256, header x-uoa-signature)	payload and confirm a briefs row appears. **Reviewers: confirm timingSafeEqual is used and unsigned requests are rejected with 401.**
View tracking	Inbound (browser → own server)	POST /api/public/track-view	None	Open any published /brief/$slug; a row should appear in brief_views. **Reviewers: confirm rate limiting before launch — currently absent (see §7).**
Google OAuth	Outbound (auth)	Lovable broker → Supabase Auth	None in app; configured in Lovable Cloud auth provider settings	“Sign in with Google” on /login should redirect through lovable.auth.signInWithOAuth('google').
Supabase Storage	Outbound	brand-assets (public), brief-media (public) buckets	Service-role for writes from server	Upload an image in advanced editor; URL should be publicly fetchable.

Security review checklist for §4

- ☐ SUPABASE_SERVICE_ROLE_KEY is referenced **only** in src/integrations/supabase/client.server.ts and never imported from anything under src/components/, src/hooks/, or any *.functions.ts that is also imported by a UI route loader.
- ☐ Every /api/public/* route validates input (Zod) and either verifies a signature (UOA) or is safe-to-be-anonymous-write (track-view, currently unrated).
- ☐ No process.env.* reads happen at module scope in shared files. Only inside .handler().
- ☐ RLS is enabled on every table in the schema. Public reads are scoped to status='published'.
- ☐ LOVABLE_API_KEY is rotated via the Lovable Cloud rotate tool, not update_secret.
- ☐ Browser bundles do not contain the strings SERVICE_ROLE or service_role — confirm with a production build search.

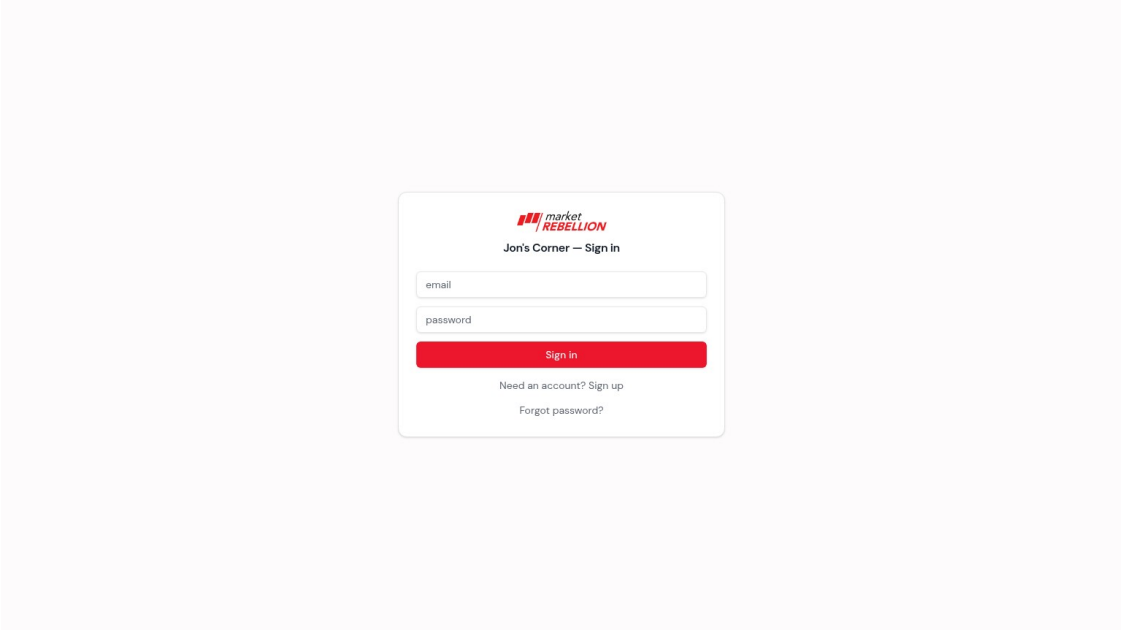
---

5. How to use it

Step-by-step operating instructions, from zero to a published brief.

5.1 Sign in

1. Go to https://jonscorner.marketrebellion.ai (production) or the preview URL.
2. Click **Sign in** (top right) — or **Open dashboard** if you want to be routed through auth.
3. Enter email and password, or click **Sign in with Google**. New users use **Need an account? Sign up**.
4. You will be redirected to /corner on success.



Sign-in page at /login. Email + password and Google OAuth.

5.2 Pick a show

5. On /corner, choose the show context: **3@3**, **Rebel's Edge**, **60 Seconds**, or **Ad-hoc**. This tags every brief in the current session.

SCREENSHOT: The /corner page with the four-up show picker visible (3@3, Rebel's Edge, 60 Seconds, Ad-hoc) before any picks are queued.

5.3 Queue picks

6. Paste a row from UOA Elite into the input — the parser (src/lib/uoa-parser.ts) extracts ticker, signal, contracts, price band.
7. Or click **Add ticker manually** and enter a symbol; the system fetches sector + logo via ticker-info.functions.ts.
8. Repeat for every pick you want in this session (typically 3–5).

SCREENSHOT: The queue step in /corner with two or three tickers added, each showing logo, symbol, signal type, and a "Remove" button.

5.4 Run the batch

9. Click **Run batch**. Behind the scenes the app calls harvest.functions.ts then brief.functions.ts for each queued ticker in sequence.
10. Watch the per-ticker status pills move through queued → harvesting → drafting → ready.

SCREENSHOT: The run step with a progress strip per ticker showing the four-state progression and a per-row spinner.

5.5 Review and edit

11. When a ticker reaches ready, click it to open the inline editor. Sections are: **Thesis**, **Bull case**, **Bear case**, **Technical setup**, **Risk parameters**, **Jon's take**.
12. Edit text directly, or click the small **regenerate** icon next to any section to re-prompt the LLM for just that section.
13. To leave the guided flow and use the dense workspace, click **Open advanced editor**. You can navigate back to /corner without losing queue state.

SCREENSHOT: The inline review editor with one brief expanded, all six section blocks visible, and a regenerate icon highlighted.

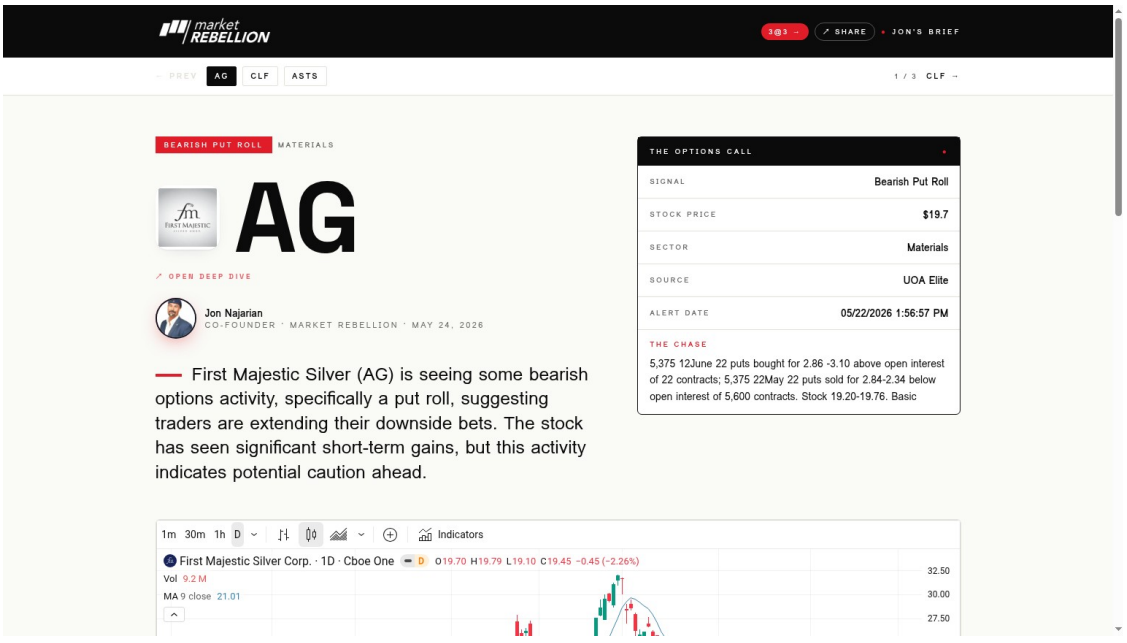
5.6 Publish

14. Click **Publish**. Choose **One page per brief** (creates /brief/$slug per pick) or **Combined show page** (groups them under one /show/$service/$date).
15. After publish, the modal shows shareable URLs. Copy and post — the page is publicly readable immediately.

SCREENSHOT: The publish modal with the two layout choices and the resulting public URLs listed.

5.7 The public brief

The reader experience for end consumers.



A published brief at /brief/brief-mpk3dtg5-1gzc. Header with ticker logo, signal badge, options call summary, Jon's portrait + thesis, and embedded TradingView chart below the fold.

5.8 Admin (staff only)

16. Staff users (those with admin or reviewer role in user_roles) can open /admin.
17. Review the analytics tile (views per brief), manage briefs (unpublish, edit metadata), and grant or revoke staff roles.

SCREENSHOT: The /admin dashboard with the analytics tile at the top, the brief list table below, and the role management drawer on the right.

---

6. File and code structure

```
src/
├── routes/                    # File-based routing
│   ├── __root.tsx             # HTML shell, providers, global onAuthStateChange
│   ├── index.tsx              # Public landing page (/)
│   ├── login.tsx              # Sign-in
│   ├── auth.tsx               # Sign-up
│   ├── auth.callback.tsx      # OAuth callback handler
│   ├── forgot-password.tsx    # Reset request
│   ├── reset-password.tsx     # New-password form
│   ├── _authenticated.tsx     # Layout guard — redirects to /login
│   ├── _authenticated/
│   │   ├── corner.tsx         # Guided + advanced workspace (Jon's main UI)
│   │   └── admin.tsx          # Staff-only dashboard
│   ├── brief.$slug.tsx        # Public brief page
│   ├── show.$service.$date.tsx  # Combined show page (per service per date)
│   ├── combined.tsx           # Combined show listing
│   └── api/
│       ├── jon-ask-stream.ts  # SSE endpoint for "Ask Jon"
│       └── public/
│           ├── uoa-webhook.ts  # Signed inbound webhook
│           └── track-view.ts   # View analytics beacon
│
├── lib/
│   ├── brief.functions.ts     # Draft / enhance / regenerate / publish a brief
│   ├── harvest.functions.ts   # Firecrawl + cache
│   ├── ingest.functions.ts    # Parse pasted UOA, normalize tickers
│   ├── slides.functions.ts    # Google Slides export
│   ├── admin.functions.ts     # Role grants, brief moderation
│   ├── ticker-info.functions.ts  # Quote / logo / profile lookups
│   ├── ai.server.ts           # Lovable AI Gateway wrapper
│   ├── ask.server.ts          # Ask Jon prompt + streaming
│   ├── chameleon.server.ts    # Jon-voice system prompt
│   ├── firecrawl.server.ts    # Firecrawl REST client
│   ├── market.server.ts       # Market data lookups
│   ├── activity.server.ts     # Audit log writer
│   ├── mr-services.ts         # Show / service enum + display config
```

```
│   ├── shows.ts              # Show definitions
│   ├── slug.ts               # Slug generation
│   ├── uoa-parser.ts         # UOA Elite row parser
│   ├── error-page.ts         # 500 page renderer
│   ├── error-capture.ts      # Frontend error sink
│   └── logger.ts             # Structured logger
│
├── components/
│   ├── guided/GuidedCorner.tsx   # The 5-step typeform-style flow
│   ├── AskJonPanel.tsx           # Streaming Q&A widget
│   ├── JonAsked.tsx              # FAQ block on /brief/$slug
│   ├── JonFaq.tsx                # Static FAQ
│   ├── TradingViewChart.tsx      # Standard TV widget
│   ├── LazyTradingViewChart.tsx  # Lazy-loaded version (perf)
│   ├── FastTradingViewChart.tsx  # Minimal sparkline variant
│   ├── TickerDetailModal.tsx     # Deep-dive modal
│   ├── TickerLogo.tsx            # Logo with fallback
│   ├── MediaAttachments.tsx      # brief-media bucket renderer
│   ├── ErrorBoundary.tsx         # React error boundary
│   ├── auth-debug-panel.tsx      # Dev-only session inspector
│   ├── logout-button.tsx
│   └── ui/                   # shadcn/ui primitives (do not edit)
│
├── hooks/
│   ├── use-auth-ready.ts         # Waits for Supabase session hydration
│   ├── use-is-staff.ts           # Calls has_role / is_staff RPC
│   ├── use-ask-jon-stream.ts     # SSE consumer for Ask Jon
│   ├── use-logout.ts
│   └── use-mobile.tsx
│
├── integrations/supabase/
│   ├── client.ts             # Browser client (publishable key) — AUTO-GENERATED
│   ├── client.server.ts          # Admin client (service role) — AUTO-GENERATED
│   ├── auth-middleware.ts        # requireSupabaseAuth — AUTO-GENERATED
│   ├── auth-attacher.ts          # Bearer header attacher — AUTO-GENERATED
│   └── types.ts              # DB types — AUTO-GENERATED
│
├── router.tsx                # createRouter + QueryClient wiring
├── start.ts                  # createStart — registers attachSupabaseAuth +
errorMiddleware
├── server.ts                 # Cloudflare Worker entry
├── styles.css                # Tailwind v4 + design tokens (oklch)
└── routeTree.gen.ts              # Auto-generated by TanStack Router plugin — DO
NOT EDIT

supabase/
├── migrations/               # Ordered SQL migrations (do not edit history)
└── config.toml               # Project-level Supabase config

wrangler.jsonc                # Cloudflare Worker config (main: src/server.ts)
```

```
vite.config.ts          # Vite + TanStack Start (entry: server.ts)
package.json              # Dependencies + scripts
```

Where to look for…

You want to understand…	Open this file
How a brief is drafted by AI	src/lib/brief.functions.ts + src/lib/ai.server.ts + src/lib/chameleon.server.ts
The Jon-voice prompt	src/lib/chameleon.server.ts
How UOA webhooks are verified	src/routes/api/public/uoa-webhook.ts
The guided 5-step flow	src/components/guided/GuidedCorner.tsx
Auth gating for protected routes	src/routes/_authenticated.tsx + src/integrations/supabase/auth-middleware.ts
Public read RLS rules	supabase/migrations/*.sql (search for create policy)
The public brief page rendering	src/routes/brief.$slug.tsx
Ask Jon streaming	src/routes/api/jon-ask-stream.ts + src/lib/ask.server.ts + src/hooks/use-ask-jon-stream.ts

Database tables

Table	Purpose
briefs	One row per pick. Holds draft + published content (thesis, bull_case, bear_case, technical_setup, risk_parameters, jons_take, media), status, public_slug, combined_slug, service, service_locked.
publications	Snapshot table for publish events.
brief_views	Per-view analytics rows written by /api/public/track-view.
brief_questions	Ask Jon Q&A pairs surfaced on a brief's FAQ block.
harvests	News harvest results per ticker.
tickers	Ticker metadata cache (logo, name, sector).
firecrawl_search_cache	Raw Firecrawl responses with TTL via updated_at.
activity_log	Internal audit trail. Staff-readable only.

Table	Purpose
user_roles	Per-user role rows (admin, reviewer). **Roles never live on a user table** — separate table prevents privilege escalation.

Database functions

Function	Purpose
has_role(_user_id, _role)	SECURITY DEFINER. Single source of truth for role checks. Used in RLS policies and from app via RPC.
is_staff(_user_id)	Returns true if user has admin OR reviewer. Used for staff gating.
set_updated_at()	Trigger helper to maintain updated_at.
handle_new_user_role()	On signup, first user becomes admin, every subsequent user becomes reviewer by default. **Reviewers: confirm this is the desired default before opening signup to the public.**

Wiring you should verify exists

- src/start.ts registers attachSupabaseAuth inside createStart's functionMiddleware. Without it, every protected server function 401s.
- src/routes/__root.tsx wires supabase.auth.onAuthStateChange once to invalidate the router + query cache.
- _authenticated/* routes never call protected server functions from a public-route loader.

---

7. Known limitations and areas to review

A QA review should pay particular attention to the items below. They are either incomplete, fragile, hardcoded, or worth a deliberate second look.

Reliability

- **No retry on AI calls.** A transient gateway error in ai.server.ts will fail an individual brief draft. The user must click regenerate manually. Add exponential-backoff with jitter (3 attempts) before launch.
- **No retry on Firecrawl.** Same shape. Cache mitigates this but a cold ticker can hard-fail.
- **Batch run is sequential.** GuidedCorner.tsx runs picks one at a time. Acceptable for ≤5 picks but doesn't scale.

- **handle_new_user_role trigger** auto-grants reviewer to every signup. This is fine while signup is invite-only — but if signup is opened to the public, this is a privilege-escalation footgun. **Flag.**

Security

- **No rate limiting** on /api/public/uoa-webhook or /api/public/track-view. The webhook is HMAC-protected; track-view is not, so it is open to flooding. Add a Cloudflare WAF rule or a Postgres-backed limiter.
- **No CSRF protection beyond TanStack defaults** on state-changing server functions. Functions are protected by requireSupabaseAuth + bearer token, which mitigates this, but it should be explicitly verified.
- **brand-assets and brief-media storage buckets are public.** Anything uploaded is world-readable by URL. Confirm no PII or unwatermarked assets land there.
- **Email templates are Supabase defaults.** Auth emails are not branded and are sent from the default Supabase sender domain. Configure a custom sender + DKIM/SPF before public launch.
- **No leaked-password (HIBP) protection** is enabled on the auth provider. One-line fix via the configure_auth tool — recommended.

Observability

- **No Sentry / external error tracking.** Frontend errors land in console + src/lib/error-capture.ts only. Worker errors live in Cloudflare logs (1-hour retention via Lovable Cloud).
- **No structured metrics.** “Briefs published per day” and “AI cost per brief” are eyeballed from raw tables, not dashboarded.
- **activity_log is written but not surfaced** anywhere in /admin. Build a viewer.

Testing

- **No automated tests.** Zero unit, zero e2e. Manual QA only. Recommended first targets: src/lib/uoa-parser.ts, src/lib/slug.ts, src/lib/mr-services.ts, then a Playwright happy-path of guided flow → publish.
- **No CI gating.** The Lovable platform runs typecheck on commit, but there is no lint/test/security-scan gate before deploy.

UX and content

- **Mobile layout** on /brief/$slug is functional but unloved. Margins, line lengths, and the TradingView height should be tuned for narrow viewports.
- **/show/$service/$date** requires both a valid service slug and a date with combined_slug matches; otherwise it 404s. The 404 page is generic — consider a “no briefs for this date” empty state.

- **Slides export (`slides.functions.ts`)** currently returns a JSON payload only. There is no renderer attached, so it is unused in production. Either wire a renderer or remove the entry point.
- **`media` column on `briefs`** is free-form JSON. No schema validation. A typo in the advanced editor can break rendering on the public brief.

Hardcoded / config-shaped values to inspect

- Show definitions are hardcoded in `src/lib/shows.ts` and `src/lib/mr-services.ts`. Adding a new show requires both a code change and a migration that updates the `service` enum / check constraint.
- The AI model choices (`google/gemini-2.5-flash`, `google/gemini-2.5-pro`, `openai/gpt-5-mini`) are referenced as string literals in `src/lib/ai.server.ts` and the server fns that call it. Consider a central `MODELS` const.
- Jon's portrait, name, and "co-founder · Market Rebellion" footer are hardcoded into `src/routes/brief.$slug.tsx`. Move to a `bylines.ts` config if more bylines are ever needed.

Day-2 ops gaps

- No runbook for "the brief queue is stuck." Recovery is: open `/admin`, find the stuck row, clear `status` back to `'queued'`. Document this.
- No soft-delete UI. Deletes go through direct DB writes via Lovable Cloud → Database.
- Secrets rotation is documented but not scripted. A rotated `UOA_WEBHOOK_SECRET` will silently start rejecting webhooks until a deploy happens — coordinate with the upstream UOA team.

End of guide. Questions or gaps? File an issue in the repo or contact the Market Rebellion engineering owner.