Launch Readiness Review — ClientNest

Sample review of an app generated with Lovable · 29 November 2026

1. Summary for the founder
2. Scope and method
3. Findings
   A. Secrets & configuration
      [LOW] A-1 · `.env` is committed and not git-ignored
   B. Database (Supabase / Row Level Security)
   C. Authentication & authorization
   D. API, server functions & third-party integrations
      [MEDIUM] D-1 · AI reminder has no per-user limit or spend cap
      [MEDIUM] D-2 · Failed payments don't downgrade the account
      [LOW] D-3 · Free-plan limit can be exceeded by parallel requests
      [LOW] D-4 · An existing Pro user can start a second subscription
      [LOW] D-5 · Webhook events aren't de-duplicated
      [LOW] D-6 · User text goes straight into the AI prompt
   E. Dependencies & build
      [LOW] E-1 · Outdated esbuild in the build toolchain
   F. Operations (backups, monitoring, cost control)
      [MEDIUM] F-1 · No error tracking or spend alerts
      [LOW] F-2 · No CI checks
   G. Code quality (maintainability)
4. What is fine
5. Recommended plan
6. Appendix

> **About this report.** ClientNest is a sample app I generated with Lovable (free plan, one prompt, November 2026) so that prospective clients can see a complete, unedited Launch Audit. It is not a client's app. Everything below comes from the same process a paid audit uses.

1. Summary for the founder

Launch Readiness Score: 81/100

ClientNest is in good shape on the basics. Every database table has Row Level Security, each user can only see their own clients and invoices, admins can't be self-appointed, and the Stripe webhook checks that events really come from Stripe. Nothing lets a stranger read or change another user's data today. The risks are in the business logic around money:

- Any signed-up user can call the AI reminder feature as often as they like, and every call is paid from your AI budget.
- A customer whose card payment fails keeps the Pro plan, because the app only downgrades someone when Stripe deletes the subscription outright.

Both take an afternoon to fix. The automated scan first scored the app 0/100 because of six "critical" alerts. I read the code behind each one: all six were false alarms, and they are listed at the end of section 4.

Severity	Count	What it means
Critical	0	Fix before anyone else uses the app. Data or money is exposed today.
High	0	Fix before launch / before the investor meeting.
Medium	3	Fix in the first month.
Low	7	Nice to have.

Top 3 actions this week

1. Put a daily per-user limit on "Write reminder email" (D-1).
2. Handle failed and changed subscriptions in the Stripe webhook, so unpaid accounts drop back to Free (D-2).
3. Turn on error tracking and a spend alert for AI usage, so you hear about problems before your users do (F-1).

2. Scope and method

- **Repository:** `chinhdq/clientnest-studio` (private), branch `main`, snapshot taken 29 November 2026.
- **Authorized by:** Chinh, owner of the sample app, on 29 November 2026.
- **Method:** static review of the source code and SQL in the repository. Automated scanners (audit-kit heuristics, semgrep, gitleaks, trufflehog, osv-scanner) ran first, then I read every flagged file and walked a 7-area manual checklist. No live systems were tested. Anything that only a dashboard can confirm is marked **verify in dashboard**.
- **Stack detected:** React + TanStack Start (server functions and server routes), Lovable Cloud (Supabase Postgres + Auth), Drizzle SQL migrations, Stripe via its REST API, Lovable AI gateway.

Limitations. This is a launch-readiness review, not a penetration test or a compliance certification. It reflects the code at the snapshot above. Hosting, DNS and third-party dashboards are out of scope unless noted.

3. Findings

Each finding says what it is, why it matters, where it is, how to fix it, and how long the fix takes.

A. Secrets & configuration

[LOW] A-1 · `.env` is committed and not git-ignored

What: `.env` is tracked in git, and `.gitignore` has no `.env` rule.

Why it matters: Today the file only holds values that are public by design: the Supabase URL, the project id and the publishable key. Lovable commits it this way by default. The risk is the next variable: if someone later adds a secret key to this file, it goes straight into git history.

Where: `.env`, `.gitignore`

Evidence: variables `SUPABASE_URL`, `SUPABASE_PROJECT_ID`, `SUPABASE_PUBLISHABLE_KEY` (and `VITE_` copies), all public values.

Fix: Add `.env*` and `!.env.example` to `.gitignore`. Keep server secrets in Lovable → Project Settings → Secrets, which the code already expects.

Effort: 10 min

B. Database (Supabase / Row Level Security)

No findings. See section 4.

Verify in dashboard: the migration in the repo matches what is deployed (Lovable Cloud → Database → Policies shows the same policies on `clients`, `invoices`, `profiles`, `subscriptions`, `user_roles`).

C. Authentication & authorization

No findings. See section 4.

Verify in dashboard:

- Email confirmation stays on.
- The redirect URL allowlist contains only your domains.
- Leaked-password protection is enabled.

D. API, server functions & third-party integrations

[MEDIUM] D-1 · AI reminder has no per-user limit or spend cap

What: The `draftReminder` server function calls the AI model every time it is invoked. Any signed-in user can invoke it, with no rate limit or daily quota.

Why it matters: Every call is paid from your AI credits. One user, or a script running on a free account, can call it thousands of times and drain the budget for everyone. Email confirmation slows this down but doesn't stop it.

Where: `src/lib/app.functions.ts:63` (`draftReminder`), `src/lib/ai.server.ts:10`

Fix:

1. Add an `ai_usage` table (user_id, day, count).
2. In `draftReminder`, refuse the call once the user is over the limit (for example 20 a day on Free and 200 on Pro).
3. Set a monthly AI spend alert in your Lovable/AI provider settings.

Effort: 2–3 h

[MEDIUM] D-2 · Failed payments don't downgrade the account

What: The webhook upgrades a user on `checkout.session.completed` and downgrades only on `customer.subscription.deleted`. It ignores `customer.subscription.updated` and `invoice.payment_failed`.

Why it matters: When a Pro customer's card fails, Stripe marks the subscription `past_due`, then `unpaid`, and retries for days or weeks before deleting it, depending on your settings. During all that time the customer keeps unlimited clients without paying. You also can't see these accounts, because the database still says `pro`.

Where: `src/routes/api/public/stripe-webhook.ts:24` and `:38`

Fix:

1. Handle `customer.subscription.updated`: set `plan = 'pro'` only while `status` is `active` or `trialing`, and set `free` otherwise.
2. Store `status` as it is today.
3. In Stripe → Webhooks, enable the two extra events.

Effort: 1–2 h

[LOW] D-3 · Free-plan limit can be exceeded by parallel requests

What: The 3-client limit is enforced by a database trigger that counts the user's rows and then inserts. Two inserts sent at the same moment can both see "2 clients" and both succeed.

Why it matters: A determined free user can end up with more than 3 clients. It costs you little, but it is a gap in the plan you charge for.

Where: `drizzle/migrations/0000_migration.sql:87` (`enforce_client_limit`)

Fix: Inside the trigger, take a per-user lock before counting: `perform pg_advisory_xact_lock(hashtext(new.user_id::text));`.

Effort: 30 min

[LOW] D-4 · An existing Pro user can start a second subscription

What: `createCheckoutSession` doesn't check whether the user is already Pro.

Why it matters: A user who clicks Upgrade twice, or reloads the success page, can end up with two monthly charges. That means a refund request, and possibly a chargeback.

Where: `src/lib/app.functions.ts:7`

Fix: Before calling Stripe, read the user's subscription and return "already on Pro" when `plan = 'pro'`. Better: send Pro users to the Stripe customer portal instead of a new checkout.

Effort: 30 min

[LOW] D-5 · Webhook events aren't de-duplicated

What: The webhook doesn't record which Stripe event ids it has already processed.

Why it matters: Stripe retries deliveries. Today the handler only sets fields, so a repeat does no harm. As soon as a "welcome to Pro" email or a credit grant is added, repeats will send or grant twice.

Where: `src/routes/api/public/stripe-webhook.ts:19`

Fix: Insert `event.id` into a `stripe_events` table with a unique constraint, and skip the event when the insert conflicts.

Effort: 1 h

[LOW] D-6 · User text goes straight into the AI prompt

What: The client name and company the user typed are placed directly into the prompt.

Why it matters: The only person affected is the user who typed it, because they see their own draft. Still, worth tightening before the AI feature can see other people's data.

Where: `src/lib/ai.server.ts:14`

Fix: Wrap user values in clear delimiters, tell the model to treat them as data, and cap their length.

Effort: 15 min

E. Dependencies & build

[LOW] E-1 · Outdated esbuild in the build toolchain

What: `esbuild@0.18.20` has a known advisory (GHSA-67mh-4wv8-2f99) affecting its development server.

Why it matters: It's a build-time tool and doesn't ship to users, so the practical risk is low. Keeping the toolchain current avoids a pile of upgrades later.

Where: `bun.lock`

Fix: Update the package that pulls in this esbuild version, then re-run `osv-scanner`.

Effort: 30 min

F. Operations (backups, monitoring, cost control)

[MEDIUM] F-1 · No error tracking or spend alerts

What: Only Lovable's built-in error reporting is present. There is no error tracker with alerts, and no budget alert on AI or Stripe.

Why it matters: When checkout or the AI feature breaks in production, you'll find out from an angry customer instead of an alert.

Where: repo-wide

Fix:

1. Add Sentry (the free tier is enough) to the client and server.
2. Alert on new errors.
3. Set spend alerts for AI usage and Lovable Cloud.

Effort: 1–2 h

[LOW] F-2 · No CI checks

What: There is no CI workflow in the repository.

Why it matters: The repo already has tests (`src/test/`), but nothing runs them automatically. A broken change can reach production unnoticed.

Where: `.github/workflows/` (missing)

Fix: Add a GitHub Actions workflow that runs typecheck, lint, the existing tests and `osv-scanner` on every push.

Effort: 1 h

Verify in dashboard: backups. Check which backup and restore option your Lovable Cloud plan includes, and restore once into a test project.

G. Code quality (maintainability)

No findings worth your money at this size. The codebase is small, typed and readable.

4. What is fine

- **Row Level Security everywhere.** Every table has RLS, with a separate policy per operation tied to the logged-in user. Invoice policies also check that the client belongs to the same user, so nobody can attach an invoice to someone else's client.
- **The plan can't be faked from the browser.** The `pro`/`free` flag lives in a `subscriptions` table that users can only read. Only the Stripe webhook, running on the server, can change it.
- **Admins can't be self-appointed.** Roles live in a separate table that users can't write to. The admin user list is checked on the server, not just hidden in the UI.
- **The Stripe webhook checks signatures properly.** It uses an HMAC over the raw body, a constant-time comparison and a 5-minute timestamp window.
- **Secrets stay on the server.** The Supabase service-role key and the Stripe secret key are only read in server code. AI calls go through the server, never from the browser.
- **Plan limits live in the database**, not just the UI, and the repo already contains tests for routing and billing.

Also checked and dismissed:

- Six scanner alerts said the Stripe webhook "does not verify signatures". It does; the scanner didn't recognise the hand-written check.
- An alert on `dangerouslySetInnerHTML` points at a standard shadcn/ui chart component that only injects static theme CSS.
- The committed `.env` was first rated Critical. It contains only public values, so it is now A-1 (Low).

5. Recommended plan

Phase	Items	Effort	Outcome
Now (before more users)	D-1 AI quota, D-2 failed-payment handling	3–5 h	No open cost or revenue leak
Before launch	D-4, D-3, D-5, F-1	3–4 h	Score ≥ 90, alerts in place
First month	A-1, D-6, E-1, F-2	2 h	Clean config, CI on every push

Fix Pack offer. I can do the "Now" and "Before launch" phases as a fixed-price engagement: **$1,500**, 3 working days. Delivery is reviewed pull requests with tests for the quota and the billing states, plus a re-scored report. 50% to start, 50% on delivery.

You do these yourself: enable the two extra webhook events in Stripe, set the spend alerts, and confirm the "verify in dashboard" items.

6. Appendix

- **Scanners:**
  - audit-kit heuristics: 10 leads.
  - semgrep: no remaining hits after removing false positives.
  - gitleaks and trufflehog: no secrets in the working tree or history.
  - osv-scanner: 1 advisory (E-1).
  - Preliminary automated score: 82/100. The score after manual verification is 81/100.
- **Machine-readable findings:** `findings.json`
- **Glossary:**
  - **RLS** (Row Level Security): database rules that decide which rows each logged-in user may read or change.
  - **Publishable / anon key:** the public Supabase key your frontend uses. It is safe only when RLS is on, as it is here.
  - **Service-role key:** the admin key. It must never reach the browser.
  - **Webhook signature:** proof that a payment event really came from Stripe.
  - **Server function:** code that runs on the server but is called from the app like a normal function. It is a public endpoint, so it needs its own auth and limits.