

This is exactly what you receive

A demonstration build made for this purpose, not a client application. The findings below are real findings in that build.

SUBJECT	Demonstration application — subscription dashboard
STACK	Lovable · React · Supabase · Postgres · Stripe · Vercel
SCOPE	Web application, database access rules, payment path, deployment configuration
ACCESS PROVIDED	Read-only: repository, Supabase dashboard (viewer), Vercel project (viewer)
TURNAROUND	Two business days from access granted
REPORT VERSION	1.0

Summary

Not safe to launch. Four critical findings, two of which expose customer data to any visitor.

The application is functionally complete and the product logic is sound. Every finding below is in the layer between a working demonstration and a live service: access control, payment correctness, and the operational configuration needed to run and recover the system. None of it requires a rebuild. We estimate the full remediation at [n] hours.

FINDINGS BY SEVERITY

4 CRITICAL	3 HIGH	3 MEDIUM	2 LOW
---------------	-----------	-------------	----------

How to read this

- Severity is about consequence, not effort. A critical finding is one that exposes data, loses money, or stops the application working for everyone.
- Every finding carries the evidence it was found from — a file and line, a configuration value, or a request we made. You can verify each one yourself.
- Effort is our estimate of the work to fix it properly, including tests. It is the basis of the sprint quote, and we do not revise it upward later.
- Findings we could not confirm with read-only access are listed as such rather than assumed either way.

Findings

PT-001 **CRITICAL**

Row-level security disabled on all public tables

EVIDENCE

supabase/migrations/ – no policy statements present. Confirmed by an unauthenticated GET to /rest/v1/profiles?select=* with the anon key from the client bundle, which returned all [n] rows.

IMPACT

Any visitor can read and write every row of every table, including user records, subscription state and payment metadata. The anon key is published in the JavaScript bundle by design, so this requires no credentials and no exploitation — only a network tab.

RECOMMENDED FIX

Enable RLS on every public table and write explicit per-role read and write policies. Add a test per table asserting that a user from one tenant cannot reach another's rows, and run it in CI so a future migration cannot silently remove isolation.

EFFORT

[n] hours

PT-002 **CRITICAL**

Service role key present in the client bundle

EVIDENCE

src/lib/db.ts:14 – SUPABASE_SERVICE_ROLE_KEY read via import.meta.env and used in a browser-executed module. Recovered from the built asset at /_astro/db.<hash>.js.

IMPACT

The service role key bypasses row-level security entirely. Anyone with the bundle has unrestricted read and write access to the database regardless of what policies are added, which means fixing PT-001 alone would leave the system fully exposed.

RECOMMENDED FIX

Rotate the key immediately. Move every call that requires elevated access behind server-side endpoints that check the caller. Add a build-time scan of client output for key patterns, failing the build on a match.

EFFORT

[n] hours

PT-003 **CRITICAL**

Stripe webhook endpoint does not verify signatures

EVIDENCE

api/webhooks/stripe.ts:8 – the handler parses the request body and acts on event.type without calling stripe.webhooks.constructEvent. No signing secret is referenced anywhere in the repository.

IMPACT

Any party who discovers the endpoint URL can post a forged event and grant themselves a paid subscription, cancel another customer's, or alter billing state. The endpoint is discoverable, because it is a conventional path.

RECOMMENDED FIX

Verify every event against the signing secret before acting on it, and reject anything that fails. Treat Stripe as the source of truth for subscription state rather than writing it from the event payload alone.

EFFORT

[n] hours

PT-004

CRITICAL

Webhook handler is not idempotent

EVIDENCE

api/webhooks/stripe.ts:22 – an insert on every checkout.session.completed event with no uniqueness constraint and no record of processed event IDs. Reproduced on a local copy by replaying one event three times, which created three subscription rows.

IMPACT

Stripe retries events on any non-2xx response and occasionally delivers duplicates in normal operation. The result is duplicate subscriptions, double entitlement and a database that disagrees with the payment provider about what the customer has paid for.

RECOMMENDED FIX

Store the Stripe event ID with a unique constraint and short-circuit on a repeat. Make subscription writes upserts keyed on the Stripe subscription ID rather than inserts.

EFFORT

[n] hours

PT-005

HIGH

Administrative actions are gated in the interface only

EVIDENCE

src/components/AdminPanel.tsx:31 hides the controls behind a role check; the corresponding endpoints in api/admin/ perform no check of their own. Confirmed on a staging deployment by calling POST /api/admin/users/<id>/role directly with a standard user session.

IMPACT

Any authenticated user can perform administrative actions by calling the endpoints directly, including changing their own role. Hiding a control is a user-experience decision, not an access control.

RECOMMENDED FIX

Enforce the role check server-side on every privileged endpoint. Keep the interface check as well, so users are not shown actions that will fail.

EFFORT

[n] hours

PT-006

HIGH

No error tracking or structured logging in production

EVIDENCE

No error reporting SDK in package.json. Server handlers catch and return a generic 500 without logging the cause; Vercel function logs show the status only.

IMPACT

Production failures are invisible. A broken payment path or a failing migration would be reported by customers rather than detected, and there would be no trace to diagnose from afterwards.

RECOMMENDED FIX

Add error tracking with release tagging and source maps, structured logging on server handlers, and an uptime check on the critical paths. Route alerts somewhere a human reads.

EFFORT

[n] hours

PT-007 HIGH

No database backups configured

EVIDENCE

Supabase project settings — point-in-time recovery disabled, no scheduled export, no restore procedure in the repository.

IMPACT

A bad migration, an accidental delete or a successful attack against PT-001 would be unrecoverable. There is no point in the system's history that could be returned to.

RECOMMENDED FIX

Enable point-in-time recovery, schedule exports with retention, and document the restore. Execute the restore once into a scratch project, because an untested backup is an assumption.

EFFORT

[n] hours

PT-008 MEDIUM

Staging and production share one database

EVIDENCE

A single SUPABASE_URL across all Vercel environments; preview deployments connect to the production project.

IMPACT

Any preview deployment can write to live customer data, and a destructive migration tested on a branch runs against production. This also means no environment exists in which a restore can be rehearsed.

RECOMMENDED FIX

Separate projects per environment with distinct credentials, and a seeded staging dataset that contains no real customer records.

EFFORT

[n] hours

PT-009 MEDIUM

No rate limiting on authentication or payment endpoints

EVIDENCE

No rate limiting middleware or platform configuration on api/auth/ or api/checkout/. On a staging deployment, [n] sequential sign-in attempts from one address were all served.

IMPACT

Credential stuffing against sign-in, and cost amplification against endpoints that call paid third-party APIs. Neither is prevented and neither would currently be noticed.

RECOMMENDED FIX

Rate limit by address and by account on authentication, and per session on endpoints with a downstream cost. Return a 429 rather than failing silently.

EFFORT

[n] hours

PT-010

MEDIUM

Server-side input validation missing on write endpoints

EVIDENCE

api/profile/update.ts:11 spreads the request body into the database update without a schema. Validation exists in the client form only.

IMPACT

Fields the form does not expose can be written directly, including any column the access rules would otherwise permit. Combined with PT-001 this is a direct path to privilege escalation.

RECOMMENDED FIX

Validate every request body against an explicit schema on the server, allow-listing fields rather than excluding known-bad ones.

EFFORT

[n] hours

PT-011

LOW

Dependencies carry known advisories

EVIDENCE

npm audit reports [n] advisories, [n] of them high, across [n] direct dependencies.

IMPACT

None currently reachable from application code, but the set grows over time and nothing is watching it.

RECOMMENDED FIX

Update the affected packages, then enable automated dependency updates with CI gating so this stays at zero without anyone remembering to look.

EFFORT

[n] hours

PT-012

LOW

Security headers not set

EVIDENCE

Response headers on the production deployment carry no Content-Security-Policy, X-Content-Type-Options or Referrer-Policy.

IMPACT

Defence in depth only — no specific vulnerability depends on it. Worth doing because it is cheap and because it constrains the blast radius of a future injection defect.

RECOMMENDED FIX

Add a content security policy and the standard header set at the edge, verified by a check in CI.

EFFORT

[n] hours

Could not be confirmed with read-only access

Listed rather than assumed. Each would be resolved in the first hours of a sprint.

- Whether the exposed service role key has already been used by a third party. This requires access to database audit logs, which are not available at the current Supabase plan.
- Whether any duplicate subscription records in production have resulted in incorrect billing. Confirming this needs a reconciliation run against Stripe with write access to report on.
- Whether historical customer data was accessed during the period PT-001 was live. Without request logs from before the audit window, this cannot be answered either way.

What we would do next

The four critical findings are the launch blockers and account for roughly half the estimated effort. We would fix those first, in that order, since PT-002 must be resolved before fixing PT-001 actually closes the exposure. The remaining findings are worth doing in the same sprint because the work overlaps, but the application could be launched safely once the criticals, PT-005 and PT-007 are closed.

INDICATIVE SPRINT

Launch Hardening Sprint, with the audit fee credited.

PutThrough - production engineering for AI-built apps. This is a sample report on a demonstration build, published at putthrough.com/sample-audit. Your report follows this format.