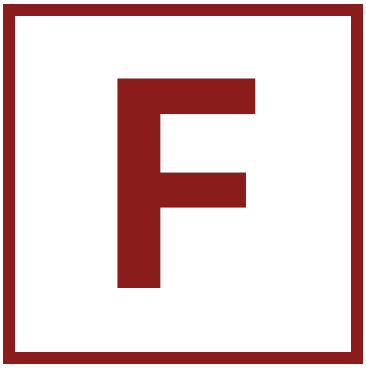
SECURITY AUDIT REPORT

Larkspur Yoga (fictional sample app)

Security and production-readiness review of the application code.



Grade F
Score 0 / 100

Prepared for	**Larkspur Yoga (fictional sample app)**
Date	**1 November 2026**
Built with	**Lovable**
Stack	**Vite · Supabase**
Code reviewed	**47 files**
Live app probed	**No, code only**

CRITICAL	HIGH	MEDIUM	LOW	INFO
6	**11**	**4**	**6**	**1**
issues, 7 places	issues, 13 places	issues	issues	issue

1 The short version

We found 27 issues to fix in Larkspur Yoga (fictional sample app): 6 critical, 11 high, 4 medium and 6 low; the most serious is “Supabase service_role key in the code”.

This key ignores every security rule of your database.

It is readable by anyone who opens your site or your repository, and whoever has it can read, change or delete the data of all your users.

16 other critical or high issues also need fixing first: “Database table open to anyone (row-level security off)”, “Stripe webhook accepts events without checking their signature” and 14 more.

Until the critical issues are fixed, assume a stranger can already do this today, with no special skill.

2 Findings

This sample shows the 3 most serious of the 27 issues to fix, in full. The full report details every issue the same way, then prices the fixes. Secrets are redacted: enough to recognise them, never enough to use them.

CRITICAL Critical

CRITICAL VA-SEC-001 Effort M: half a day

Supabase service_role key in the code

WHERE

src/lib/supabaseAdmin.ts:5 — legacy JWT key, role=service_role

```
eyJhbG…NH1n
```

WHAT IT MEANS

This key ignores every security rule of your database. It is readable by anyone who opens your site or your repository, and whoever has it can read, change or delete the data of all your users.

HOW TO FIX

1. In Supabase, open Project Settings → API and **roll** the service_role / secret key (the old one stops working).
2. Remove the key from the code and from the git history.
3. Move the code that needed it into a Supabase Edge Function, where the key is read from a server-side secret (`Deno.env.get("SUPABASE_SERVICE_ROLE_KEY")`).
4. In the browser, only ever use the anon / publishable key.

CRITICAL VA-DB-001 Effort M: half a day

Database table open to anyone (row-level security off)

WHERE

supabase/migrations/20250916093047_9d2f6b31-0c4e-4a8b-b7e5-1f3a6c9d2e84.sql:2 — public.payments: row-level security never enabled

```
CREATE TABLE public.payments ( id UUID PRIMARY KEY DEFAULT gen_random_uuid(), user_id UUID REFERENCES auth.users(id), email TEXT, stripe_session_id TEXT UNIQUE, amount_cents INTEGER NOT NULL, pack TE…
```

WHAT IT MEANS

Your app talks to its database straight from the browser, with a public key that anyone can copy from your site. This table has no access rules switched on, so anyone with that key can download, change or delete every row in it, without even logging in.

HOW TO FIX

1. Add a migration that switches row-level security on:

```
alter table public.<table> enable row level security;
```

2. Add one policy per action the app needs, tied to the signed-in user, e.g.

```
create policy "Owners read their rows" on public.<table>
  for select to authenticated using ((select auth.uid()) = user_id);
```

3. Never disable row-level security in a later migration to "make it work": add the missing policy instead.
4. Check in Supabase → Database → Tables that the table shows "RLS enabled".

CRITICAL VA-PAY-001 Effort S: under 1 hour

Stripe webhook accepts events without checking their signature

WHERE

supabase/functions/stripe-webhook/index.ts:12 — handles checkout.session.completed but never verifies the Stripe signature

```
case "checkout.session.completed": {
```

WHAT IT MEANS

Your app unlocks paid features when it receives a "payment succeeded" message, but it never checks that the message really comes from Stripe. Anyone can send a fake one and get the paid plan, credits or orders for free.

HOW TO FIX

1. In Stripe → Developers → Webhooks, copy the endpoint's signing secret and store it as a server-side secret (`STRIPE_WEBHOOK_SECRET`).
2. Verify every event with the raw request body before using it:

```
const signature = req.headers.get("stripe-signature")!;
const body = await req.text();
const event = await stripe.webhooks.constructEventAsync(
  body, signature, Deno.env.get("STRIPE_WEBHOOK_SECRET")!,
);
```

3. Answer 400 when verification fails, and never parse the body with `req.json()` first.
4. Send a test event from the Stripe dashboard to check it still works.

3 What we checked

43 checks run: 15 passed, 27 failed, 1 noted for information. 1 more could not run on this project, as noted below.

Secrets and keys 9 checks

× Failed	VA-SEC-001	Supabase service_role key in the code	critical
× Failed	VA-SEC-002	Secret in a browser-exposed environment variable	critical
✓ Passed	VA-SEC-003	Payment or cloud secret key in the code	critical
× Failed	VA-SEC-004	AI provider key in the code	critical
✓ Passed	VA-SEC-005	Email or SMS provider key in the code	high
✓ Passed	VA-SEC-006	Stripe test secret key in the code	low
× Failed	VA-SEC-007	.env file with secrets saved in the repository	high
× Failed	VA-SEC-008	Secret still readable in the git history	high
✓ Passed	VA-SEC-009	Google Maps key in the browser: check its restrictions	low

Database and file storage 15 checks

× Failed	VA-DB-001	Database table open to anyone (row-level security off)	critical
✓ Passed	VA-DB-002	Anyone can add, change or delete rows (policy allows everything)	critical
× Failed	VA-DB-003	Personal data readable by anyone	high
i Noted	VA-DB-004	Table readable by anyone	info
× Failed	VA-DB-005	Any signed-in user can reach everyone's personal data	high
× Failed	VA-DB-006	Users can give themselves admin rights, credits or a paid plan	high
× Failed	VA-DB-007	Privileged database function without a fixed search_path	medium
✓ Passed	VA-DB-008	Privileged database function callable by anyone	high
× Failed	VA-DB-009	Users can upload, overwrite or delete other people's files	high
× Failed	VA-DB-010	Storage bucket for private-looking files is public	medium
✓ Passed	VA-DB-011	Database view bypasses row-level security	high
✓ Passed	VA-DB-012	Database rules are not in the repository	medium
✓ Passed	VA-DB-020	Firebase database open to everyone	critical
✓ Passed	VA-DB-021	Any signed-in user can reach every Firebase document	high
✓ Passed	VA-DB-022	Firebase Storage writable by anyone or by any signed-in user	high

Sign-in and access control 6 checks

× Failed	VA-AUTH-001	Edge Function callable by anyone, with no check	high
× Failed	VA-AUTH-002	Permissions decided from data the user can edit	high

× Failed	VA-AUTH-003	Admin area protected only in the browser	high
✓ Passed	VA-AUTH-004	Edge Function acts on any user id sent by the caller	medium
× Failed	VA-AUTH-005	Email confirmation is turned off	low
× Failed	VA-AUTH-006	Edge Function accepts calls from any website	low

Payments 3 checks

× Failed	VA-PAY-001	Stripe webhook accepts events without checking their signature	critical
✓ Passed	VA-PAY-002	Payment amount taken from the request	high
× Failed	VA-PAY-003	Paid status written from the browser	high

Web page safety 4 checks

× Failed	VA-WEB-001	Unsanitised HTML inserted into the page	medium
✓ Passed	VA-WEB-002	Code built from data and run with eval or new Function	medium
✓ Passed	VA-WEB-003	Redirect to any address given in the link	low
× Failed	VA-WEB-004	Production source maps enabled	low

Cost abuse (AI and paid APIs) 2 checks

× Failed	VA-COST-001	AI service called straight from the browser	critical
× Failed	VA-COST-002	Paid API endpoint open to anyone, with no rate limit	high

Dependencies 1 check

– Not run	VA-DEP-001	Dependency with a known critical or high vulnerability Not part of this audit (option --deps).	high

Production readiness 4 checks

× Failed	VA-PROD-001	.gitignore does not keep .env files out of the repository	medium
× Failed	VA-PROD-002	No automated tests	low
× Failed	VA-PROD-003	No error boundary: any crash shows a blank page	low
× Failed	VA-PROD-004	Session, token or password printed to the browser console	low

The live app was not probed. With the owner's written consent, read-only probes confirm what a stranger can reach today: readable tables, listable storage, keys in the shipped JavaScript, exposed files.

4 Method, scope and consent

What was read	The source code of the project folder: 47 files (Lovable, Vite · Supabase). Installed dependencies, build output, lockfiles and binary files were skipped. Nothing was changed.
Live probes	Not run: this report covers the code only.
Evidence	Every secret shown in this report is redacted. The full machine-readable list of findings is in findings.json.
Severity	• **Critical**: the data of all users readable or writable, money or keys exposed. • **High**: exploitable by a motivated stranger. • **Medium**: exploitable under some conditions, or a serious risk of outage. • **Low**: hardening and good practice. • **Info**: context for the reader, not counted in the grade.
Grade	The score starts at 100 and loses 30 points per critical issue, 12 per high, 5 per medium and 1 per low, counted once per issue. A from 90, B from 75, C from 60, D from 40, F below 40. Any critical issue caps the grade at D; user data or a key readable on the live app makes it F.
Tool	vibe-audit 0.1.0, run on 1 November 2026 at 10:09 UTC.

An audit lowers risk; it is not a guarantee that no other flaw exists.