

SAMPLE REPORT. TaskNest is a fictional app that Postlaunch built with deliberate flaws to demonstrate an audit. No real company, customer or data is involved. The access-control and performance results in Appendix A are real: we ran the tests against TaskNest's actual schema.

Production-Readiness Audit

TaskNest: team task manager (React + Supabase + Stripe)

PREPARED FOR
TaskNest (fictional), Founder

REPORT DATE
21 October 2026

TESTING WINDOW
14–18 October 2026

PREPARED BY
Postlaunch

REPORT ID
PL-SAMPLE-0001

CLASSIFICATION
Confidential: client only

1. Executive summary

Verdict: not ready for more customers or payments. Three issues let anyone on the internet read customer billing data, take full control of the database, or upgrade accounts to Pro without paying. All three can be fixed in about two days. Until they are, every new signup adds to the exposure.

3

Critical

4

High

5

Medium

3

Low

What this means in plain English

- Your customers' billing details are public.** Anyone can download every invoice (names, addresses, amounts) without logging in. (PL-01)
- The master key to your database is in your website code.** Anyone who opens the browser's developer tools can read, change or delete everything. (PL-02)
- Pro upgrades can be faked.** Your payment handler believes any message that claims "this person paid". (PL-03)
- Customers can see each other's work.** Any signed-in user can read and edit every other company's tasks and add themselves to any workspace. Your dashboard says "RLS enabled", but the rules allow everything. (PL-04)

Do these today, even if you don't hire us (about 1 hour): rotate the Supabase service-role key and the OpenAI key; turn on RLS for `invoices` ; take the admin page offline. Section 5 has the full plan.

Scorecard

Area	Grade	Main reason
Data access control	F	Table without RLS; policies that allow everyone (PL-01, PL-04, PL-05)
Secrets & keys	F	Service-role and OpenAI keys shipped to the browser (PL-02, PL-06)
Payments	F	Unsigned webhook trusted (PL-03)
Privacy	D	All emails visible; public file bucket (PL-08, PL-09)
Reliability & recovery	D	No tested restore; no monitoring (PL-10, PL-13)
Performance	C	Missing indexes; about 7× slower task list at scale (PL-11)
Email deliverability	D	No DMARC; sending service not authorized in SPF (PL-12)
Maintainability & process	D	AI edits deploy straight to production (PL-14)

Target after the Stabilize Sprint: A or B in every area. We re-test and re-grade after the fixes.

2. Scope, authorization and method

Authorization

Testing was carried out under a written *Authorization to Test* signed by the TaskNest founder on 13 October 2026. It names the systems, test accounts and dates below. Postlaunch does not test any system without written authorization.

In scope	Out of scope
Web app at <code>app.tasknest.app</code> and its staging copy Supabase project (database, RLS, Auth, Storage, Edge Functions), read-only dashboard access Source repository (read access) Stripe test mode and webhook configuration DNS and email-sending configuration for <code>tasknest.app</code>	Denial-of-service or load testing against production Social engineering of staff Third-party platforms themselves (Supabase, Stripe, Vercel, OpenAI) Attacks against real customer accounts. We used two test accounts created for the audit.

Method

- **Two-user access testing.** We signed in as two unrelated customers ("Alice" at Acme, "Bob" at Bolt) and as an anonymous visitor, then tried to read and change each other's data at the database level (Appendix A).
- **Source and bundle review.** We read every RLS policy, Edge Function and environment variable, and inspected the built JavaScript bundle for secrets.
- **Payment flow review.** We checked the Stripe webhook against Stripe's signature verification requirements.
- **Configuration review.** Auth settings, storage buckets, backups, monitoring, deployment process, DNS (SPF/DKIM/DMARC).
- **Performance check.** We loaded a staging copy with 1,000,000 tasks across 2,000 workspaces and timed the main task-list query.

Limitations

This is a point-in-time review of the systems listed above. It reduces risk but cannot prove the absence of all vulnerabilities. Changes made after the testing window are not covered. We recommend a re-test after fixes and after any major feature.

Severity definitions

CRITICAL	Exploitable now by anyone on the internet, with direct loss of data, money or control.
HIGH	Exploitable by any signed-in user, or leads to financial loss or account takeover.
MEDIUM	Weakens privacy, reliability or performance; exploitable only in combination with something else or at scale.
LOW	Hygiene and process gaps that make future incidents more likely or harder to recover from.

3. Findings at a glance

ID	Finding	Severity	Effort
PL-01	Invoices readable by anyone on the internet	CRITICAL	0.5 h
PL-02	Master database key shipped to every visitor	CRITICAL	3 h
PL-03	Anyone can upgrade any account to Pro for free	CRITICAL	2 h
PL-04	Any signed-in user can read and edit every customer's tasks	HIGH	3 h
PL-05	Users can make themselves admin and give themselves Pro	HIGH	1 h
PL-06	OpenAI key exposed in the browser	HIGH	2 h
PL-07	Admin page loads every user's data before checking who you are	HIGH	2 h
PL-08	Every user can see every other user's name and email	MEDIUM	1 h
PL-09	File attachments are public and never expire	MEDIUM	2 h
PL-10	No tested backup or restore	MEDIUM	2 h
PL-11	Task list slows as data grows (missing indexes)	MEDIUM	0.5 h
PL-12	Password-reset and receipt emails at risk of spam or rejection	MEDIUM	1 h + rollout
PL-13	No error monitoring or alerting	LOW	1.5 h
PL-14	AI edits deploy straight to production	LOW	3 h
PL-15	Weak sign-up and abuse protection	LOW	1 h

Effort is our estimate for the fix alone. Re-testing and documentation are included in the Stabilize Sprint price (Section 5).

4. Detailed findings

PL-01 Invoices readable by anyone on the internet

CRITICAL

Where	<code>public.invoices</code> , defined in <code>supabase/schema.sql</code>
What we found	Row-Level Security was never turned on for <code>invoices</code> . Supabase gives the public <code>anon</code> role access to tables by default, and the anon key ships with every page load. So anyone can query the table directly, with no login needed.
Why it matters	Every customer's name, billing address and payment amounts can be downloaded by a stranger. That is a reportable personal-data breach under GDPR/UK GDPR and many US state laws.
Evidence	Test T1: an anonymous request returned 1 of 1 invoice rows (Appendix A).

```
-- schema.sql: RLS enabled on four tables, but not on invoices
alter table public.tasks enable row level security;
-- (no "alter table public.invoices enable row level security")
```

Fix	Enable RLS, revoke anon access, and let users read only their own invoices. Writes happen only from the payment webhook (server side).
Verified	After the fix, T1 returns <i>permission denied</i> ; Alice still sees her own invoice (T8).

PL-02 Master database key shipped to every visitor

CRITICAL

Where	<code>src/lib/supabase.ts</code> , <code>.env</code>
What we found	The Supabase service-role key is read from <code>VITE_SUPABASE_SERVICE_ROLE_KEY</code> . Vite copies every <code>VITE_*</code> variable into the public JavaScript bundle. The service-role key ignores all security rules.
Why it matters	Anyone can open developer tools, copy the key, and read, change or delete the entire database, including user accounts. No other fix matters until this key is rotated.

```
// src/lib/supabase.ts
const serviceKey = import.meta.env.VITE_SUPABASE_SERVICE_ROLE_KEY;
export const supabaseAdmin = createClient(url, serviceKey);
```

Fix	(1) Rotate the key in the Supabase dashboard today. (2) Delete <code>supabaseAdmin</code> from the front end. (3) Move admin operations into an Edge Function that checks on the server that the caller is an admin. (4) Search git history and old deployments for the leaked key.
-----	---

PL-03 **Anyone can upgrade any account to Pro for free**

CRITICAL

- Where** `supabase/functions/stripe-webhook/index.ts`
- What we found** The webhook reads the request body and acts on it without checking Stripe's signature. The endpoint URL follows Supabase's standard pattern, so it is easy to guess.
- Why it matters** Anyone can send a hand-written "checkout completed" message and get Pro for free, or change the plan of any user whose ID they know. This is lost revenue and a way to tamper with accounts.

```
Deno.serve(async (req) => {  
  const event = await req.json(); // trusted without verification  
  if (event.type === "checkout.session.completed") { ... update({ plan: "pro" }) ... }  
})
```

- Fix** Verify every event with `stripe.webhooks.constructEventAsync` and the webhook signing secret. Act only when `payment_status === "paid"`. Record processed `event.id` s so retries can't double-apply. Replacement code is ready (`fixes/stripe-webhook.ts`).
- Verified** Code review. Confirm in staging that an unsigned request gets `400` and `stripe trigger checkout.session.completed` succeeds.

PL-04 **Any signed-in user can read and edit every customer's tasks**

HIGH

- Where** RLS policies on `tasks` and `workspace_members`
- What we found** The policies exist, so Supabase shows "RLS enabled", but they use `using (true)`, which allows everything. Any user can also insert themselves into any workspace. This is the same class of issue as CVE-2025-48757, which affected about 10% of scanned Lovable apps.
- Why it matters** One customer can read another company's confidential plans and change their tasks. For a B2B product this ends trust, and enterprise deals with it.
- Evidence** T2: Bob (Bolt) read **2 of 2** of Acme's tasks, including one titled "Q4 pricing change" with notes "Confidential: raise Pro to \$49". T3: Bob added himself to Acme's workspace.

```
create policy "tasks readable" on public.tasks  
  for select to authenticated using (true);  
create policy "members join" on public.workspace_members  
  for insert to authenticated with check (true);
```

- Fix** Policies based on membership through an `is_workspace_member()` function; only workspace owners can add members (`fixes/001_security_fixes.sql`).
- Verified** After the fix: T2 returns 0 rows, T3 is blocked, and Alice still sees her own tasks (T6).

PL-05 **Users can make themselves admin and give themselves Pro**

HIGH

Where	Update policy on <code>profiles</code>
What we found	Users may update their own profile row, but nothing limits <i>which columns</i> they change. <code>role</code> and <code>plan</code> sit in the same row.
Evidence	T4: Bob ran one update and came back as <code>role = admin, plan = pro</code> .
Fix	Allow updates to <code>full_name</code> only (a column-level grant) and add <code>with check . role</code> and <code>plan</code> change only from server-side code.
Verified	After the fix, T4 is denied and Alice can still rename herself (T7).

PL-06 **OpenAI key exposed in the browser**

HIGH

Where	<code>src/lib/ai.ts</code>
What we found	The task-summary feature calls OpenAI straight from the browser, using <code>VITE_OPENAI_API_KEY</code> .
Why it matters	The key is visible in the network tab. Leaked AI keys are collected by automated scanners and used to run up bills, often within hours.
Fix	Rotate the key. Move the call into an authenticated Edge Function (<code>fixes/ai-summarize.ts</code>) that reads only tasks the user is allowed to see, caps output length and is rate-limited. Set a monthly spending limit in the OpenAI dashboard.

PL-07 **Admin page loads every user's data before checking who you are**

HIGH

Where	<code>src/pages/Admin.tsx</code>
What we found	The page fetches all profiles with the service-role client <i>before</i> it checks whether the visitor is an admin. The redirect hides the page, but the data has already reached the browser.

```
useEffect(() => { supabaseAdmin.from("profiles").select("*")... }, []);  
if (profile?.role !== "admin") return <Navigate to="/" />; // too late
```

Fix	Admin data is served only by a server-side function that checks the caller's role in the database. The front-end route guard stays, but only for convenience. Shares work with PL-02.
-----	---

PL-08 **Every user can see every other user's name and email**

MEDIUM

What we found The `profiles` read policy is `using (true)`. T5: Bob read Alice's email address.

Why it matters It's a privacy breach, gives competitors your customer list, and makes targeted phishing easy.

Fix Users can read their own profile and the profiles of people they share a workspace with. Verified: T5 returns 0 rows after the fix.

PL-09 **File attachments are public and never expire**

MEDIUM

What we found The `attachments` storage bucket is public. Anyone with a file link (in a forwarded email, browser history or logs) can download it, and links never expire.

Fix Make the bucket private, store files under the workspace ID, add a storage policy for members, and serve files through signed URLs that expire after 60 minutes. Requires small app changes.

PL-10 **No tested backup or restore**

MEDIUM

What we found No point-in-time recovery, no copy of the data stored anywhere else, and no restore has ever been tried. All editors can, and sometimes do, generate destructive database migrations.

Fix Turn on point-in-time recovery (plan upgrade), add a nightly encrypted `pg_dump` to separate storage, and run a documented restore drill each quarter.

PL-11 **Task list slows as data grows**

MEDIUM

What we found There are no indexes on `tasks.workspace_id` or `workspace_members.user_id`. The security checks on each row hit these columns too.

Evidence On a staging copy with **1,000,000 tasks**, the task-list query took a median of **~104 ms** without indexes and **~14 ms** with them, about 7× faster (5 runs each, Appendix A). The gap grows with more data and more simultaneous users.

Fix Three indexes, included in the fix migration. No app changes needed.

PL-12 **Password-reset and receipt emails at risk of spam or rejection**

MEDIUM

What we found `tasknest.app` has no DMARC record, and the SPF record for the Resend sending subdomain is missing. Since November 2025 Gmail rejects or junks mail from senders that aren't authenticated properly, and Outlook.com has similar rules.

```
$ dig +short TXT _dmarc.tasknest.app
(no record)
$ dig +short TXT send.tasknest.app
(no record) # Resend expects an SPF record here
```

Fix Add the SPF and DKIM records Resend shows in its dashboard. Publish DMARC at `p=none` with reporting, move to `quarantine` after two clean weeks, then to `reject`.

PL-13 **No error monitoring or alerting**

LOW

What we found Errors are invisible unless a customer complains. Failed webhooks and Edge Function errors go unnoticed.

Fix Sentry (free tier) for front end and functions, an uptime check on the app and webhook, and alerts to email or Slack.

PL-14 **AI edits deploy straight to production**

LOW

What we found There is no staging environment and no review step. Every AI-generated change, including database changes, goes live immediately, and nothing in version control records those database changes.

Fix A GitHub repo with branch previews, a separate staging Supabase project, and database changes kept as migration files that are reviewed before they're applied.

PL-15 **Weak sign-up and abuse protection**

LOW

What we found Email confirmation is off, the minimum password length is 6, there's no CAPTCHA on sign-up, and there's no rate limit on the AI feature.

Fix Turn on confirmation, require 10+ character passwords, add Turnstile or hCaptcha on sign-up, and limit each user to a set number of AI requests per hour.

5. Fix plan and fixed price

Phase	When	Findings	Outcome
0. Stop the bleeding	Today (about 1 h)	Rotate keys (PL-02, PL-06), RLS on invoices (PL-01), admin page offline (PL-07)	Nothing critical left exposed to anonymous visitors
1. Lock down	Days 1–3	PL-02 to PL-07	Customers isolated from each other; payments verified; no secrets in the browser
2. Harden	Week 2	PL-08 to PL-12	Privacy, files, backups, speed and email deliverability fixed
3. Keep it fixed	Ongoing	PL-13 to PL-15, plus a monthly re-check	Problems caught before customers notice them

Stabilize Sprint: fixed price

Includes	Price
Fixes for all 15 findings · fix migration applied to staging and then production · re-run of every access test · updated report with new grades · 45-minute handover call and written runbook	\$4,600 fixed 50% to start, 50% on delivery About 2 weeks
Optional: Care Plan Essential . Monitoring, dependency and security updates, monthly security re-check and report, 2 hours of changes each month	\$290 / month Cancel anytime

You own all code, migrations and documentation. Fixes are delivered as reviewed changes in your repository, so you can apply them yourself or with another developer if you prefer.

Appendix A: Verification results

Each test ran at the database level as a real Supabase role (`anon` or `authenticated` with a user's ID), inside a transaction that was rolled back afterwards. Tests were run on TaskNest's schema before and after the fix migration. The test scripts are included with the fix package (`tests/20_access_tests.sql`).

Test	What we tried	Before fix	After fix
T1	Anonymous visitor reads invoices	1 row returned	Permission denied
T2	Bob reads Acme's tasks	2 rows returned	0 rows
T3	Bob adds himself to Acme's workspace	Succeeded	Blocked by row-level security
T4	Bob sets himself to admin + Pro	Succeeded (admin, pro)	Permission denied
T5	Bob reads Alice's email	alice@acme.test	0 rows
T6	Alice reads her own tasks (must still work)	2 rows	2 rows
T7	Alice renames herself (must still work)	Succeeded	Succeeded
T8	Alice reads her own invoice (must still work)	1 row	1 row

Performance (PL-11)

Dataset	Query	Without indexes	With indexes
1,000,000 tasks · 2,000 workspaces · ~6,000 memberships	Latest 50 tasks for one workspace, as a signed-in member, with fixed RLS	median 104 ms (88–390 ms, 5 runs)	median 14 ms (13–26 ms, 5 runs)

Environment: PostgreSQL 16 in a local container with Supabase's roles and defaults recreated. Absolute timings on Supabase hosting will differ; the relative improvement is what matters.

Appendix B: Checklist coverage

Area	Check	Result
Access control	RLS enabled on every table in exposed schemas	Fail (PL-01)
	Policies restrict by owner or membership (no using (true) on private data)	Fail (PL-04, PL-08)
	INSERT/UPDATE policies have with check	Fail (PL-04, PL-05)
	Privileged columns (role, plan, credits) not writable by users	Fail (PL-05)
	Two-user and anonymous access tests pass	Fail (Appendix A)
Secrets	No service-role or secret keys in the client bundle	Fail (PL-02, PL-06)
	No secrets in git history	Fail (rotate after PL-02)
	Third-party API calls made from the server side	Fail (PL-06)
Auth	Admin checks enforced on the server	Fail (PL-07)
	Email confirmation, password policy, bot protection	Fail (PL-15)
	OAuth redirect URLs restricted to own domains	Pass
Payments	Webhook signatures verified	Fail (PL-03)
	Plan and entitlements set only by the server	Fail (PL-03, PL-05)
	Live and test keys separated per environment	Pass
Storage	Private buckets for user files	Fail (PL-09)
	Signed URLs with expiry	Fail (PL-09)
Reliability	Backups enabled and restore tested	Fail (PL-10)
	Error monitoring and alerts	Fail (PL-13)
	HTTPS everywhere, security headers	Pass
Performance	Indexes on foreign keys and columns used by RLS	Fail (PL-11)
	Lists paginated	Pass
Email	SPF and DKIM for every sending service	Fail (PL-12)
	DMARC published and moving toward enforcement	Fail (PL-12)
Process	Staging environment and reviewed migrations	Fail (PL-14)
	Dependencies without known critical vulnerabilities	Pass