

Security and Launch Review

App reviewed: TaskNest, a multi-tenant project tracker (Next.js, Supabase, Stripe)

Review type: Code and configuration review before go-live

Date: October 2026

About this sample. TaskNest is a demo app written by Omega Matrix for this sample, with common launch mistakes left in on purpose. It is not a client project and it was never deployed. A real review follows exactly this format, and every finding in a real review is confirmed against the client's own code and settings.

1. Summary

Launch verdict: not ready. As built, any visitor could read every customer's invoices, and any signed-in user could read or change other companies' data, add themselves to another company as an admin, or upgrade to a paid plan for free. All of these can be fixed in about **2 to 3 working days**, and none needs a rebuild.

3

Critical

4

High

3

Medium

2

Low

Fix these three before anything else:

1. Take the Supabase service role key out of the browser and rotate it (F1).
2. Turn on row level security for the invoices table and make the invoices storage bucket private (F2, F7).
3. Verify Stripe webhook signatures (F3).

2. Scope and method

Covered	Database schema and row level security policies, Supabase storage settings, edge functions, Next.js API routes, client-side Supabase usage, auth flows, environment and secret handling, payment webhooks.
Method	Manual review of the repository by a senior engineer, checked against the Supabase security guidance and the OWASP API Security Top 10. Each finding lists the file and line, what can go wrong, the fix and the effort.
Not covered	Load testing, a full penetration test of live systems, and infrastructure outside the repository. Nothing is tested on a live system without written permission.

3. Findings at a glance

ID	Severity	Finding	Where	Effort
F1	CRITICAL	Service role key shipped to the browser	src/lib/supabase.ts:8	2 hours
F2	CRITICAL	Invoices table has no row level security	001_init.sql:33	1 hour
F3	CRITICAL	Stripe webhook accepts unsigned events	api/stripe/webhook/route.ts:5	1 hour
F4	HIGH	Any user can read and edit every company's projects	001_init.sql:39, 42	2 hours

F5	HIGH	Any user can add themselves to any company	001_init.sql:48	1 hour
F6	HIGH	Invite function trusts company and role from the request	functions/invite-member/index.ts:4, 12	3 hours
F7	HIGH	Invoice files stored in a public bucket	001_init.sql:50	2 hours
F8	MEDIUM	Full customer list readable by any user	001_init.sql:36	30 minutes
F9	MEDIUM	Admin page protected only by a browser setting	src/app/admin/page.tsx:7	2 hours
F10	MEDIUM	Password reset accepts any redirect address	src/app/reset/page.tsx:6	1 hour
F11	LOW	Secret file not excluded from git	.gitignore	15 minutes
F12	LOW	Backups, monitoring and rate limits not confirmed	Project settings	Half a day

4. Findings in detail

CRITICAL F1. Service role key shipped to the browser

Where: `.env.local:3` and `src/lib/supabase.ts:8`

What we found: The service role key is stored as `NEXT_PUBLIC_SUPABASE_SERVICE_ROLE_KEY`. Next.js puts every `NEXT_PUBLIC_` variable into the JavaScript sent to the browser, and the shared client uses this key whenever it is present.

What can go wrong: The service role key skips all row level security. Anyone who opens the browser developer tools can copy it and read, change or delete every row in the database. This is the most common critical issue in AI-built Supabase apps.

Fix: Keep the service role key on the server only, use the anon key in the browser, and rotate the exposed key in the Supabase dashboard.

```
# .env.local
SUPABASE_SERVICE_ROLE_KEY=...      # no NEXT_PUBLIC_ prefix, server only

// src/lib/supabase.ts (browser)
export const supabase = createClient(url, process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!)

// src/lib/supabase-admin.ts (server routes only)
export const supabaseAdmin = createClient(url, process.env.SUPABASE_SERVICE_ROLE_KEY!)
```

Note: the anon key is meant to be public. It is safe only when row level security is correct, which is why F2, F4, F5 and F8 matter.

CRITICAL F2. Invoices table has no row level security

Where: `supabase/migrations/001_init.sql:33`

What we found: Row level security is turned on for three tables but left for later on invoices.

What can go wrong: Supabase exposes every table through its API. Without row level security, anyone holding the public anon key can download every invoice, including customer email addresses and amounts, for every company on the platform.

Fix:

```
alter table invoices enable row level security;

create policy "org members read invoices" on invoices
  for select using (
    org_id in (select org_id from members where user_id = auth.uid())
  );
```

CRITICAL F3. Stripe webhook accepts unsigned events

Where: `src/app/api/stripe/webhook/route.ts:5`; `STRIPE_WEBHOOK_SECRET` is empty in `.env.local:5`

What we found: The route reads the request body as JSON and upgrades the company to the pro plan without checking that the request came from Stripe.

What can go wrong: Anyone can send a fake payment event and upgrade any company for free.

Fix:

```
const body = await req.text()
const sig = req.headers.get('stripe-signature')!
const event = stripe.webhooks.constructEvent(body, sig, process.env.STRIPE_WEBHOOK_SECRET!)
```

HIGH F4. Any user can read and edit every company's projects

Where: `001_init.sql:39` and `:42`

What we found: The project policies only check that the user is signed in. They do not check that the user belongs to the company that owns the project.

What can go wrong: One customer can see and change another customer's projects. In a multi-tenant app this is the leak that ends B2B deals.

Fix: Check company membership in both `using` and `with` check.

```
create policy "members read projects" on projects for select
  using (org_id in (select org_id from members where user_id = auth.uid()));

create policy "members update projects" on projects for update
  using (org_id in (select org_id from members where user_id = auth.uid()))
  with check (org_id in (select org_id from members where user_id = auth.uid()));
```

HIGH F5. Any user can add themselves to any company

Where: `001_init.sql:48`

What we found: The insert policy on members allows any row, including any `org_id` and any `role`.

What can go wrong: A signed-in user can join another company as an admin and take it over.

Fix: Remove the insert policy. Add members only through a server function that checks the inviter is an admin of that company (see F6).

HIGH F6. Invite function trusts company and role from the request

Where: `supabase/functions/invite-member/index.ts:4`, `:12` and `:15`

What we found: The function takes `org_id` and `role` from the request body and writes them with the service role key. It never checks who is calling, and it allows calls from any website.

What can go wrong: Any signed-in user can invite anyone, including themselves, into any company with any role. This is the API flaw OWASP ranks first, broken object level authorization.

Fix: Read the caller from their token, confirm they are an admin of `org_id`, allow only known roles, and limit allowed origins to your own domain.

HIGH F7. Invoice files stored in a public bucket

Where: `001_init.sql:50`

What we found: The invoices storage bucket is created as public.

What can go wrong: Anyone with a file link can download an invoice PDF, with no sign-in and no expiry.

Fix: Make the bucket private, add storage policies by company, and serve files through short-lived signed URLs.

MEDIUM F8. Full customer list readable by any user

Where: `001_init.sql:36`

What can go wrong: The organisations policy is using `(true)`, so any signed-in user can list every customer company and its plan. Competitors can sign up and read your customer list.

Fix: Limit reads to companies the user belongs to.

MEDIUM F9. Admin page protected only by a browser setting

Where: `src/app/admin/page.tsx:7`

What can go wrong: Admin access is decided by a value in the browser's local storage, which any user can set. Combined with F1, this page shows every company.

Fix: Decide admin rights on the server or in database policies, using the `members.role` column, never in the browser.

MEDIUM F10. Password reset accepts any redirect address

Where: `src/app/reset/page.tsx:6` and `:7`

What can go wrong: The reset link sends users to whatever address is in the `next` parameter. An attacker can send a genuine-looking reset email that lands on their own site.

Fix: Accept only paths inside your app, and keep the Supabase Redirect URLs allow list limited to your own domains.

LOW F11. Secret file not excluded from git

Where: `.gitignore`

Fix: Add `.env*.local` to `.gitignore`. If the file was ever committed, rotate every key in it.

LOW F12. Backups, monitoring and rate limits not confirmed

What we found: These live in project settings, not in code, so they need confirming with the team before launch.

Fix: Confirm daily backups or point-in-time recovery, error monitoring with alerts, a staging project for testing changes, and auth rate limits.

5. Launch readiness checklist

Check	Result
Row level security on every table exposed through the API	Fail (F2)
Policies check company membership, not just sign-in	Fail (F4, F5, F8)
Service role key used only on the server	Fail (F1)
Private storage for customer files	Fail (F7)
Payment webhooks verified	Fail (F3)
Server-side checks on every function that writes data	Fail (F6)
Admin rights decided on the server	Fail (F9)
Auth redirects limited to own domains	Fail (F10)
Secrets kept out of git	Fail (F11)
Anon key used correctly in the browser	Pass, once F1 is fixed
Backups, monitoring, staging, rate limits	To confirm (F12)

6. What happens after the review

Every client gets this report, the fixes ranked by risk and effort, and a 30-minute call to walk through it. The team can fix the issues themselves, or Omega Matrix can fix them in a fixed-price sprint and then re-check every finding.

Omega Matrix Software Solutions, Chennai, India

Independent security and launch reviews for apps built with Supabase, Lovable, Bolt, Replit and Cursor
vasantha@omegamatrix.in | omegamatrix.in