IsMyVibeGood

Deep Code Audit

yourname/my-saas-app

June 25, 2026

TECH STACK	Next.js 14 (App Router) + React, TypeScript, Tailwind CSS on the frontend. API routes in Next, Supabase (Postgres) for data and auth, Stripe for payments, deployed on Vercel.
WHAT IT IS	A subscription SaaS where small businesses create, schedule, and track social posts. Visitors sign up, pick a plan, connect their accounts, and manage posts from a dashboard.
HOW IT FLOWS	Browser (React) calls Next API routes, which read and write Supabase. Stripe Checkout handles payment and a webhook updates the user plan. Scheduled posts are picked up by a cron route that calls the social APIs.

16 / 100

Grade F

1 critical
6 high
11 medium
3 low

Executive summary

The product works and the core flows are in place, but it is not yet safe to put real customers and their data on. Three things are urgent: a secret key is exposed to the browser, a database query can be manipulated by visitors, and any logged-in user can read other accounts' data. Beyond security, the app loads slowly on phones and weak connections, has no pagination so it will get worse as data grows, and has no automated checks to stop regressions. None of these are hard to fix, and every finding below includes a prompt you can paste straight into your AI tool. Fix the security items first, then work down the action plan.

Security 5 findings

The biggest risks live here. There is a secret exposed to the browser, an injectable query, and a missing ownership check. Treat the first three as launch-blockers.

CRITICAL

Secret key exposed to the browser

lib/supabase.ts:4

Your Supabase service-role key is stored in a NEXT_PUBLIC_ variable, which means it is bundled into the JavaScript every visitor downloads. Anyone can open dev tools, copy it, and read or delete your entire database.

YOUR CODE

```
   2  import { createClient } from '@supabase/supabase-js'
   3
   4  export const supabase = createClient(url,
process.env.NEXT_PUBLIC_SUPABASE_SERVICE_ROLE_KEY!)
```

Fix: Move the service-role key to a server-only env var (no NEXT_PUBLIC_ prefix), use it only in API routes, and rotate the exposed key immediately. The browser should only ever see the anon/public key.

PROMPT TO FIX IT

```
My Supabase service-role key is in a NEXT_PUBLIC_ env var and shipped to the client
in lib/supabase.ts. Move it to a server-only variable used only in API routes,
switch the client to the anon key, and show me every file I need to change.
```

HIGH

SQL injection in the search endpoint

app/api/search/route.ts:18-25

The search query glues the visitor's text directly into the SQL string. Someone can type special characters to read or delete data they should never reach.

YOUR CODE

```
  18  const q = req.nextUrl.searchParams.get('q')
  19  const { rows } = await db.query(
  20    `SELECT * FROM posts WHERE title LIKE '%${q}%'`
  21  )
  22  return Response.json(rows)
```

Fix: Use parameterized queries or the Supabase query builder so user input is always treated as data, never as SQL.

PROMPT TO FIX IT

```
Rewrite the query in app/api/search/route.ts to use parameterized queries instead of
string concatenation, so user input cannot inject SQL. Keep the same behavior and
explain the change.
```

HIGH

Any logged-in user can read any account

app/api/posts/[id]/route.ts:10-16

The endpoint loads a post by id but never checks it belongs to the person asking. Changing the id in the URL returns another customer's data.

YOUR CODE

```
10  export async function GET(req, { params }) {
11    const post = await db.post.findUnique({
12      where: { id: params.id }
13    })
14    return Response.json(post)
15  }
```

Fix: After loading the record, confirm its owner matches the logged-in user and return 404 if not. Apply this to every per-user route.

PROMPT TO FIX IT

```
In app/api/posts/[id]/route.ts, add an ownership check so a user can only access
their own posts, returning 404 otherwise. Then scan my other API routes for the same
missing check and list them.
```

MEDIUM

No rate limiting on login or signup

app/api/auth/route.ts

Nothing slows repeated attempts, so an attacker can try thousands of passwords per minute against your users.

Fix: Add per-IP rate limiting with a short lockout after several failed attempts.

PROMPT TO FIX IT

```
Add per-IP rate limiting to my auth routes with a lockout after repeated failures.
Suggest a lightweight approach that works on Vercel and show the implementation.
```

MEDIUM

Outdated dependency with a known vulnerability

package.json

One or more packages are several versions behind and have publicly documented security issues, which attackers actively scan for.

Fix: Update the flagged packages and turn on automated dependency alerts.

PROMPT TO FIX IT

```
Run npm audit on my project, update the packages with known vulnerabilities without
breaking anything, and set up Dependabot so I am alerted to future issues.
```

Code quality & architecture 2 findings

The code is readable but repeats itself and has no automated formatting, which will slow you down as it grows. No spaghetti yet, but a few patterns are worth tightening now.

MEDIUM

The same data-fetching logic is copied across components

components/*

Several components each write their own fetch + loading + error code. When you change how data loads, you have to edit it in many places and they will drift.

Fix: Extract the shared fetching into a single reusable hook (for example useApi) and have components call it.

PROMPT TO FIX IT

```
I have duplicated fetch, loading, and error logic across several components. Create one reusable hook that handles fetching, loading, and errors, and refactor the components to use it.
```

LOW

No linter or formatter configured

There is no ESLint/Prettier setup, so formatting is inconsistent and easy mistakes are not caught automatically.

Fix: Add ESLint + Prettier with a shared config and a format script, then format the codebase once.

PROMPT TO FIX IT

```
Set up ESLint and Prettier for a Next.js + TypeScript project with sensible defaults, add format and lint scripts to package.json, and tell me the commands to format everything once.
```

Performance 2 findings

On a fast laptop this feels fine, but on phones and weak connections it is slow. The main issues are a heavy initial download and unoptimized images.

HIGH

The whole app loads up front (no code splitting)

app/dashboard/page.tsx

Charts and editor libraries are imported into the first page load, so even visitors who never open them download everything. On a phone or slow connection the app takes many seconds to become usable.

Fix: Lazy-load heavy components (charts, editors) with dynamic imports so they only download when actually used.

PROMPT TO FIX IT

```
In my Next.js dashboard, the chart and editor libraries load on first paint. Convert them to dynamic imports with next/dynamic so they only load when needed, and point out other heavy imports I should split.
```

MEDIUM

Images are full-size and unoptimized

components/PostCard.tsx:12

Images use a plain img tag at full resolution, so a phone downloads desktop-sized files. This wastes data and slows rendering on older devices.

YOUR CODE

```
11  return (
12    <img src={post.imageUrl} className="card-img" />
13  )
```

Fix: Use the Next.js Image component (or responsive sizes + modern formats) so images are resized and lazy-loaded.

PROMPT TO FIX IT

```
Replace the plain img tags in components/PostCard.tsx with the Next.js Image
component, set proper width/height and sizes, and explain the benefit for mobile
users.
```

Scalability & data 2 findings

This works at small scale but will degrade as customers add data. Lists load everything at once and some screens lack loading and error states.

HIGH

List endpoints return every row (no pagination)

app/api/posts/route.ts:8

The posts endpoint returns all rows in one response. A customer with thousands of posts will see slow loads and high memory use, and it gets worse over time.

YOUR CODE

```
 7  export async function GET() {
 8    const posts = await db.post.findMany()
 9    return Response.json(posts)
10  }
```

Fix: Add pagination (limit + offset or cursor) to list endpoints and load more on scroll or with pages.

PROMPT TO FIX IT

```
Add cursor-based pagination to app/api/posts/route.ts and update the dashboard list
to load pages instead of all rows at once. Show both the API and the UI changes.
```

MEDIUM

Missing loading and error states

app/dashboard/page.tsx

Some screens show nothing while data loads and break silently on failure, which feels broken to users on slow connections.

Fix: Add explicit loading skeletons and friendly error states with a retry option.

PROMPT TO FIX IT

```
Add loading skeletons and clear error states (with retry) to my dashboard data
fetching. Use a consistent pattern I can reuse across pages.
```

CI/CD & deployment 1 finding

Deploys happen automatically through Vercel, which is good, but there are no automated checks before code ships, so a bad change can reach customers.

MEDIUM

No CI checks run before deploy

.github/

There is no GitHub Actions workflow, so type errors, lint problems, and failing tests are not caught before code goes live.

Fix: Add a GitHub Actions workflow that runs typecheck, lint, and tests on every pull request.

PROMPT TO FIX IT

```
Create a GitHub Actions workflow for a Next.js + TypeScript app that runs install,
typecheck, lint, and tests on every pull request, and fails the check if any step
fails.
```

Monitoring & operations 3 findings

If something breaks in production right now, you will hear it from a customer, not your tools. There is no error tracking, no alerting, and no backups configured.

HIGH

No error tracking or alerting

When the app throws an error for a real user, nothing records it and nobody is notified. You only find out if a customer complains, and you cannot see how often it happens.

Fix: Add an error-tracking service (for example Sentry) on the frontend and the API, and route alerts to email or Slack.

PROMPT TO FIX IT

```
Add Sentry error tracking to my Next.js app on both the client and the API routes,
capture unhandled errors, send me alerts, and show me where to put the DSN safely.
```

HIGH

No database backups configured

If data is deleted or corrupted there is no way to restore it. For a paid product holding customer data, this is a serious risk.

Fix: Turn on automated daily backups (Supabase offers this) and do one test restore so you know it works.

PROMPT TO FIX IT

```
Walk me through enabling automated daily backups for my Supabase Postgres database
and how to do a test restore so I know my data is recoverable.
```

MEDIUM

Third-party failures are not handled gracefully

app/api/*

When Stripe or a social API is slow or down, requests hang or crash with a raw error instead of failing cleanly, which can take a page down for everyone.

Fix: Add timeouts, safe retries, and friendly fallback responses around external calls.

PROMPT TO FIX IT

```
Add timeouts and graceful error handling around my external API calls (Stripe,
social APIs) so a slow or failing third party returns a clean error instead of
hanging or crashing the page.
```

Customer journey & UX 3 findings

The flow is understandable but loses people in a few places: the pricing page is hard to use on mobile, search engines see almost nothing, and checkout lacks reassurance.

MEDIUM

Pricing page is not usable on mobile

app/pricing/page.tsx

The pricing table overflows the screen on phones, so the most important buying page is hard to read where a lot of visitors are.

Fix: Make the pricing layout responsive (stack the tiers on small screens) and test at common phone widths.

PROMPT TO FIX IT

```
My pricing table overflows on mobile. Make app/pricing/page.tsx fully responsive by stacking the tiers on small screens with Tailwind, and tell me the breakpoints you used.
```

MEDIUM

No SEO metadata

app/layout.tsx

Pages have no titles, descriptions, or social preview tags, so search engines and shared links show almost nothing, costing you free traffic.

Fix: Add per-page metadata (title, description, Open Graph) using Next.js metadata exports.

PROMPT TO FIX IT

```
Add proper SEO metadata to my Next.js app: a default title/description in app/layout.tsx and per-page metadata exports for the main pages, including Open Graph tags for link previews.
```

LOW

Form fields are missing labels (accessibility)

components/SignupForm.tsx

Inputs rely on placeholder text with no real labels, which is hard for screen readers and for anyone once they start typing.

Fix: Add associated label elements for every input.

PROMPT TO FIX IT

```
Add accessible, associated labels to every input in components/SignupForm.tsx without changing the visual design much, and note any other accessibility quick wins you see.
```

Legal & privacy 2 findings

You collect emails and take payments but have none of the basic legal pages or consent in place. These are low-effort to add and expected by customers and payment providers.

MEDIUM

No privacy policy or terms of service

The site collects personal data and payment details but has no privacy policy or terms. This erodes trust, and Stripe and most platforms expect them.

Fix: Add a privacy policy and terms of service and link them in the footer and at signup.

PROMPT TO FIX IT

```
Generate a starter privacy policy and terms of service for a SaaS that collects emails and takes Stripe payments, then add the linked pages and footer links to my Next.js app.
```

LOW

Analytics runs without consent

app/layout.tsx

Analytics loads before the visitor agrees, which is a problem under GDPR for visitors in the EU and UK.

Fix: Add a simple consent banner that loads non-essential tracking only after the visitor agrees.

PROMPT TO FIX IT

```
Add a lightweight cookie consent banner to my Next.js app that only loads analytics
after the user accepts and remembers their choice.
```

Building on it with AI 1 finding

You are building with AI tools, but there is no shared reference that tells the AI your conventions, so each new feature risks reintroducing the issues above. A short rules document fixes most of that.

MEDIUM

No conventions doc for AI-assisted development

Without a written set of rules (folder structure, where secrets live, how to query the database, required loading/error states, ownership checks), your AI tool will keep guessing and produce inconsistent, sometimes insecure code on each feature.

Fix: Create a short conventions/architecture document and reference it at the start of every feature request so new code follows the same safe patterns.

PROMPT TO FIX IT

```
Based on this codebase, write a CONVENTIONS.md I can give my AI tool at the start of
every feature: the folder structure, where env secrets may and may not go, how to
query the database safely, when to add pagination, required loading and error
states, and the ownership check every per-user API route needs. Keep it concise and
copy-pasteable.
```

Your action plan

Do these in order. Start at the top.

1. **Move the service-role key off the client and rotate it** — SMALL EFFORT
   It is currently downloadable by any visitor and gives full database access. This is the single highest risk.
2. **Fix the SQL injection and add the ownership check** — SMALL EFFORT
   Both let outsiders reach data they should never see. Quick changes, huge risk reduction.
3. **Turn on database backups and error tracking** — SMALL EFFORT
   So you can recover from data loss and learn about failures before your customers do.
4. **Add the privacy policy and terms** — SMALL EFFORT
   You collect personal data and take payments; customers and Stripe expect these, and they are quick to add.
5. **Add pagination and loading/error states to lists** — MEDIUM EFFORT
   Prevents slowdowns as customers add data and makes the app feel reliable on weak connections.
6. **Split heavy code and optimize images** — MEDIUM EFFORT
   Cuts first-load time dramatically for mobile and older devices, where you lose the most users.
7. **Add a CI workflow and a conventions doc** — MEDIUM EFFORT
   Stops regressions before they ship and keeps every future AI-built feature consistent and safe.