Repo Review: qs-fixture-demo

Tier: starter · Repository: `/tmp/qs-fixture-demo` · Generated 2026-09-14 02:10 UTC · Quiet Shift

Read this first. This is a static review of the repository as submitted. It is not a penetration test and not a SOC 2, HIPAA, or PCI certification. Findings are best-effort. Compliance is a legal determination made by a court or regulator, not a scanner. The reviewed copy of the repository has been deleted; only this report is kept, for 30 days.

Summary

26 / 100

Critical 2 · High 2 · Medium 1

5 verified findings. Score = 100 − 25 per critical − 10 per high − 4 per medium − 1 per low, floored at 0. It measures what was found, not what is safe.

This fixture app leaks a live-looking Stripe key and a Supabase service-role key straight from a committed .env file, and its Stripe webhook will fulfil orders for anyone who POSTs to it without any signature check.

This is a minimal 4-file Next.js app, but the few lines it has concentrate serious risk: real-shaped payment and database admin credentials are checked into .env, one of those credentials (service_role) is wired into a component with no server-only boundary, the only Postgres table has no RLS policies, and the Stripe webhook trusts an unsigned request body to decide what order to fulfil for how much money. The single biggest risk is the committed secrets combined with the unsigned webhook — together they mean both database bypass and forged payment fulfilment are possible without needing to break anything, just to read or POST to what's already public.

What we looked at

Framework	Next.js, React
Auth	Supabase Auth
Database	Supabase (Postgres)
Payments	Stripe
AI SDKs	OpenAI
Deploy target	Vercel
API routes / server actions	2 / 0
Size	30 lines of app code in 4 files

What was covered

The review always walks seven areas, whether or not an automated tool pointed at them. For each area: what was checked and the files that were read.

Area	Status	Notes
Authorization	Covered	Only 2 route handlers and 1 client component exist; no server actions. chat/route.ts and webhook/route.ts have no session/auth check, and AdminTable.tsx builds a privileged Supabase client with no gating on who can render it. No RLS policies exist to backstop missing app-level checks (see F3). `app/api/chat/route.ts`, `app/api/webhook/route.ts`, `components/AdminTable.tsx`, `supabase/migrations/0001_init.sql`
Secrets	Covered	Committed .env contains a live-looking Stripe secret key and a Supabase service_role JWT plus a NEXT_PUBLIC_-prefixed 'secret' token that ships to the browser by design; service_role key is also referenced from a non-server-marked component. `.env`, `.env.example`, `components/AdminTable.tsx`
Payments	Covered	Stripe webhook handler reads req.json() directly and trusts body.type/metadata.orderId/amount_total with no stripe.webhooks.constructEvent signature check and no idempotency guard before calling fulfil(). `app/api/webhook/route.ts`, `package.json`
AI endpoints	Covered	chat/route.ts calls openai.chat.completions.create with no auth check, no per-user quota, and no rate limiting; messages array is currently empty in this fixture but the route as written accepts no validation before it would be wired to request input. `app/api/chat/route.ts`
Rate limiting and abuse	Covered	No signup/login/password-reset endpoints exist in this app. The only sensitive endpoints (chat LLM call, Stripe webhook) have no rate limiting or throttling of any kind. `app/api/chat/route.ts`, `app/api/webhook/route.ts`
Data exposure	Covered	No select('*') or query calls exist in the 30 lines of app code reviewed. Primary data-exposure risk is structural: profiles table (email, stripe_customer_id) has RLS disabled and no policies (see F3), and admin Supabase client is built in a component with no server marker (see F2). `supabase/migrations/0001_init.sql`, `components/AdminTable.tsx`
Dependencies	Covered	package.json pins next 14.2.3, react 18.3.1, @supabase/supabase-js 2.45.0, stripe 16.2.0, openai 4.56.0. Automated npm_audit run produced no candidate findings surfaced to this review; did not independently verify CVE databases. `package.json`, `package-lock.json`

Automated tools

- Ran: builtin, semgrep, gitleaks, npm_audit
- Skipped: pip_audit (no requirements.txt at the repo root)

What was not covered

- Could not verify actual Postgres grants (GRANT/REVOKE on anon/authenticated roles) since only one migration file exists and no seed/grants files were found — RLS-absence risk in F3 is based on the migration alone.
- No CLAUDE.md, README, or comment-based instructions were found in the repository to assess for prompt-injection content; nothing to flag.
- Anything only visible at runtime: live behaviour, deployed config, database contents.
- Git history beyond the shallow clone (gitleaks sees depth-1 unless a full clone is given).
- Dependencies not pinned in a lockfile at the repo root.

Findings

Critical (2)

F1 · Live-looking Stripe secret key and Supabase service_role JWT committed to .env

Critical · Secrets · fix effort: minutes · confidence: high · `.env:1`

Evidence

Line 1: STRIPE_SECRET_KEY=sk_live_…redacted ; Line 3: NEXT_PUBLIC_API_SECRET_TOKEN=hunter2hunter2hunter2 ; Line 4: SUPABASE_SERVICE_ROLE_KEY=eyJhbGci...role":"service_role"... all committed in a tracked .env file (gitleaks confirms it's in git history, commit 02024f7c).

Impact

Anyone with read access to the repo (or its git history, even after later 'fixing' the working tree) gets a Stripe secret key capable of creating charges/refunds/payouts, a Supabase service_role key that bypasses all RLS and can read/write every table, and a token whose name implies it authorizes privileged API calls. The NEXT_PUBLIC_ prefix on the third value also means it is bundled into client JS regardless of where it's committed.

Fix

Rotate all three credentials immediately in the Stripe and Supabase dashboards, remove .env from git tracking and history, and load secrets only from the deploy platform's env store.

```
git rm --cached .env
echo '.env' >> .gitignore
# rotate STRIPE_SECRET_KEY and SUPABASE_SERVICE_ROLE_KEY in their dashboards
# then purge history: git filter-repo --path .env --invert-paths
```

F2 · Supabase service_role key wired into a React component with no server-only marker

Critical · Secrets · fix effort: minutes · confidence: high · `components/AdminTable.tsx:5`

Evidence

createClient(process.env.NEXT_PUBLIC_SUPABASE_URL!, process.env.SUPABASE_SERVICE_ROLE_KEY!) is called directly inside `export function AdminTable()` (lines 4-9), which has no 'use server' directive, no 'server-only' import, and returns JSX — the shape of a component Next.js can render on the client.

Impact

If this component (or anything importing it) is ever rendered client-side or included in a client bundle, the RLS-bypassing service_role key ships to every visitor's browser, giving them full read/write access to the database regardless of RLS policies.

Fix

Move the service-role client into a server-only module and never construct it in a component file that can be part of the client bundle.

```
import 'server-only';
import { createClient } from '@supabase/supabase-js';

export function getAdminClient() {
  return createClient(process.env.NEXT_PUBLIC_SUPABASE_URL!,
process.env.SUPABASE_SERVICE_ROLE_KEY!);
}
// call getAdminClient() only from a Server Component or Route Handler, never
export it to client code
```

High (2)

F4 · Stripe webhook accepts unsigned, unauthenticated payloads and fulfils orders without idempotency

High · Payments · fix effort: hours · confidence: high · `app/api/webhook/route.ts:8`

Evidence

`const body = await req.json();` (line 8) is parsed with no call to stripe.webhooks.constructEvent, no reading of the `stripe-signature` header, and no dedupe/idempotency check before `await fulfil(body.data.object.metadata.orderId, body.data.object.amount_total)` (line 10) runs on every POST whose body.type equals 'checkout.session.completed'.

Impact

Anyone who can reach this public URL can POST a crafted JSON body to trigger order fulfilment for an arbitrary orderId and amount_total of their choosing — effectively free/forged order fulfilment with no Stripe involvement at all. Even legitimate Stripe retries would double-fulfil the same order since there is no idempotency check.

Fix

Verify the Stripe signature on the raw body before trusting the event, and record processed event ids to make fulfilment idempotent.

```
const sig = req.headers.get('stripe-signature')!;
const rawBody = await req.text();
const event = stripe.webhooks.constructEvent(rawBody, sig,
process.env.STRIPE_WEBHOOK_SECRET!);
if (await alreadyProcessed(event.id)) return new Response('ok');
if (event.type === 'checkout.session.completed') {
  await fulfil(event.data.object.metadata.orderId,
event.data.object.amount_total);
  await markProcessed(event.id);
}
```

F3 · No row-level security on public.profiles despite storing email and Stripe customer id

High · Authorization · fix effort: hours · confidence: medium · `supabase/migrations/0001_init.sql:2`

Evidence

The only migration creates `public.profiles (id uuid primary key, email text, stripe_customer_id text)` (lines 2-6) with no ALTER TABLE ... ENABLE ROW LEVEL SECURITY and no CREATE POLICY statement anywhere in supabase/migrations.

Impact

With RLS disabled, any role granted table access under Postgres/Supabase defaults (commonly including the anon/authenticated roles used by the public NEXT_PUBLIC_SUPABASE_URL client) can read or modify every user's email and Stripe customer id, enabling cross-tenant data exposure without needing the leaked service_role key at all.

Fix

Enable RLS on profiles and add owner-scoped policies before this table is queried from any client-exposed context.

```
alter table public.profiles enable row level security;
create policy "select own profile" on public.profiles
  for select using (auth.uid() = id);
create policy "update own profile" on public.profiles
  for update using (auth.uid() = id);
```

Medium (1)

F5 · Chat route calls OpenAI with no authentication, authorization, or rate limiting

Medium · AI endpoints · fix effort: hours · confidence: medium · `app/api/chat/route.ts:2`

Evidence

`export async function POST(req: Request) { const openai = new OpenAI(); const r = await openai.chat.completions.create({model:'gpt-4o',messages:[]}); return Response.json(r); }` (lines 2-5) has no session check (no Supabase auth import/call), no per-user quota, and no rate-limit logic anywhere in the file.

Impact

As written, any caller who can reach this endpoint can trigger billed gpt-4o completions with no per-user limit, enabling unmetered spend/abuse; once request input is wired into `messages`, this same lack of gating would also be the path for unauthenticated prompt injection.

Fix

Require an authenticated Supabase session and apply per-user/IP rate limiting before calling OpenAI.

```
const { data: { user } } = await supabase.auth.getUser();
if (!user) return new Response('unauthorized', { status: 401 });
const { success } = await ratelimit.limit(user.id); // e.g. Upstash ratelimit
if (!success) return new Response('rate limited', { status: 429 });
```

Remediation order

Quick wins on the worst problems first.

1. F1 Live-looking Stripe secret key and Supabase service_role JWT committed to .env (critical, minutes)
2. F2 Supabase service_role key wired into a React component with no server-only marker (critical, minutes)
3. F4 Stripe webhook accepts unsigned, unauthenticated payloads and fulfils orders without idempotency (high, hours)
4. F3 No row-level security on public.profiles despite storing email and Stripe customer id (high, hours)
5. F5 Chat route calls OpenAI with no authentication, authorization, or rate limiting (medium, hours)

Appendix: automated tool output

Automated candidates only. These are not findings. Step 3 of the method (verified review) reads the code and decides reachable/not before anything reaches the customer's report. 12 candidate(s) were produced; the findings above are the ones the verified review confirmed.

Dismissed candidates (4)

- `builtin:review_hint_route_without_auth_marker` at `app/api/webhook/route.ts:1`: Webhooks are legitimately called by Stripe's servers without a user session; the correct control is signature verification, not user auth — tracked separately as F4.
- `builtin:env_file_committed` at `.env:1`: Folded into F1 (live secrets committed in .env) rather than reported as a separate finding, per one-finding-per-root-cause; same lines are not duplicated across findings.
- `semgrep:generic.secrets.security.detected-stripe-api-key.detected-stripe-api-key` at `.env:1`: Same underlying secret as F1 (Stripe live key); not a separate root cause.
- `gitleaks:jwt` at `.env:4`: Same underlying secret as F1 (Supabase service_role JWT); not a separate root cause, though it does corroborate that the key is present in git history (commit 02024f7c), not just the working tree.

All candidates (12)

Severity	Tool	Rule	Location	Message
critical	builtin	`stripe_live_secret_key`	`.env:1`	Stripe live secret key in the repository. `STRIPE_SECRET_KEY=sk_live_…redacted`
critical	builtin	`supabase_service_role_jwt`	`.env:4`	JWT with role=service_role committed. This key bypasses row level security. `SUPABASE_SERVICE_ROLE_KEY=eyJhbGci…redacted`
critical	builtin	`service_role_key_in_client_dir`	`components/AdminTable.tsx:7`	Service-role key referenced in a client-side path with no 'use server' / 'server-only' marker. If this file reaches the browser bundle the key is public. `process.env.SUPABASE_SERVICE_ROLE_KEY!,`
high	builtin	`env_file_committed`	`.env:1`	Environment file is committed to the repository. Even if the current values are dummies, check git history for real ones and rotate anything that was ever committed.
high	builtin	`public_env_var_names_secret`	`.env:3`	NEXT_PUBLIC_API_SECRET_TOKEN is exposed to the browser by the bundler and its name suggests a secret.

				`NEXT_PUBLIC_API_SECRET_TOKEN=hunter2hunter2hunter2`
high	builtin	`review_hint_webhook_no_signature_check`	`app/api/webhook/route.ts:1`	Webhook handler with no signature-verification words in the file (constructEvent / signature / hmac / svix). Verify step: check whether the payload is trusted unsigned, and whether it is idempotent.
high	builtin	`review_hint_no_rls_policies`	`supabase/migrations:1`	1 migration file(s) and no 'enable row level security' or 'create policy' statement found. Verify step: are tables reachable with the anon key?
high	gitleaks	`jwt`	`.env:4`	Uncovered a JSON Web Token, which may lead to unauthorized access to web applications and sensitive user data. (commit 02024f7c, author T) `REDACTED`
high	semgrep	`generic.secrets.security.detected-stripe-api-key.detected-stripe-api-key`	`.env:1`	Stripe API Key detected `requires login`
medium	builtin	`review_hint_route_without_auth_marker`	`app/api/chat/route.ts:1`	API route with no auth-looking import or call. Verify step: is it callable unauthenticated, and does it take an id it does not own-check (IDOR)?
medium	builtin	`review_hint_llm_route_unmetered`	`app/api/chat/route.ts:1`	Route calls an LLM with no rate-limit/quota words in the file. Verify step: unauthenticated or unmetered LLM spend, and prompt-injection paths to tools.
medium	builtin	`review_hint_route_without_auth_marker`	`app/api/webhook/route.ts:1`	API route with no auth-looking import or call. Verify step: is it callable unauthenticated, and does it take an id it does not own-check (IDOR)?

Quiet Shift is built and operated by an AI agent with a human owner who reviews daily. Questions or a re-check: support@quietshift.dev.

Method: inventory from lockfiles and config; automated pass (Semgrep, gitleaks, dependency audits, built-in checks); verified review in which every candidate and every checklist area is read in context and only what is reachable is reported; then this write-up. https://quietshift.dev