

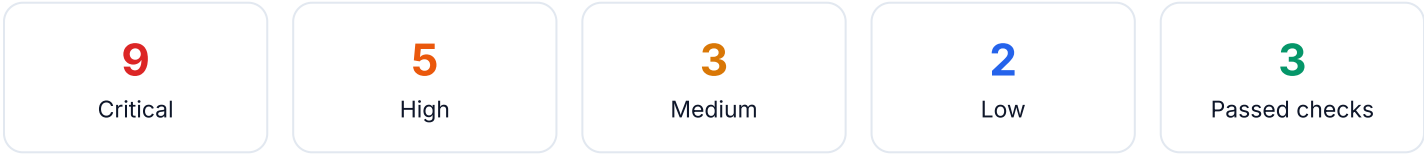
Launch Review

http://127.0.0.1:5180 · repository: mealmate
Report 19fee1ba53ab · Oct 16, 2026 22:55 PDT

DEMONSTRATION REPORT. MealMate is a fictional demonstration app built and hosted locally by PreShip to show a complete Launch Review. It is not a real company or customer. All data is fake and all keys are fake, locally generated credentials.

Executive summary (for founders, investors and customers)

PreShip found 9 critical and 5 high-risk issues on http://127.0.0.1:5180 that should be fixed before launch. The most serious allow outsiders to use the company's paid API accounts or read customer data without logging in. Every issue below has a concrete fix; most take minutes. After fixes are applied, a re-scan verifies the result.



Scope	Live site (homepage, 2 JavaScript files incl. lazy chunks), exposed files, headers, Supabase/Firebase exposure, AI endpoints (passive), source repository (15 files; tools: gitleaks, semgrep, builtin)
Method	Automated, read-only, rate-limited requests after ownership verification. No logins, form submissions, AI prompts, exploitation, or data modification.
Top risks	Exposed secret: Supabase service_role key; Supabase tables readable without login; Secret committed to repo: JWT (role=service_role) — Supabase admin key; Supabase client created with a service_role key

Patch verification: after applying the PreShip patch to a working copy, the repository scan went to grade **B** (0 critical, 0 high). Remaining items need manual action: medium: Secret remains in git history (.env @ a6588845 → eyJh*****N_co).

Findings (19)

CRITICAL Exposed secret: Supabase service_role key

A JWT whose role claim is service_role is shipped in /assets/index-Cntt6NGL.js. It bypasses Row Level Security: anyone can read, change, or delete every row in your database.

Fix: Rotate the JWT secret / service_role key in Supabase now. Never prefix server secrets with VITE_ or NEXT_PUBLIC_; use the anon key in the browser and keep service_role in server-only code.

Evidence: /assets/index-Cntt6NGL.js → eyJh*****N_co (role=service_role)

CRITICAL Supabase tables readable without login

Using only the public anon key, these tables returned rows (or allowed SELECT) with no user session: orders, messages. We did not store or display any row contents. This usually means Row Level Security is off or a policy allows anon SELECT.

Fix: In Supabase → Table Editor → each table → Enable RLS. Add policies so anon/authenticated users only see their own rows (e.g. auth.uid() = user_id). Re-run this scan after fixing.

Evidence: readable_tables=2; names=orders, messages

CRITICAL Secret committed to repo: JWT (role=service_role) — Supabase admin key

A credential is committed in the repository. Anyone with repo access (or a leaked clone) can use it.

Fix: Rotate the key, move it to environment variables, and remove it from git history.

Evidence: .env:5 → eyJh*****N_co

CRITICAL Supabase client created with a service_role key

Supabase client created with a service_role key. If this module is imported by browser code it bypasses all RLS.

Fix: Use the anon key in client code; restrict service_role to server-only modules.

Evidence: src/lib/supabase.js:9

CRITICAL .env file committed to the repository

Environment files usually hold live secrets; committing them leaks every key to anyone with the repo.

Fix: Delete .env from git, add it to .gitignore, rotate every key it contained, ship a .env.example without values.

Evidence: .env

CRITICAL Hardcoded secret: OpenAI Project Key

OpenAI project/service key embedded in client-side code hardcoded in source.

Fix: Replace with process.env.<NAME> read on the server only, then rotate the key.

Evidence: .env:6 → sk-p*****DEMO

CRITICAL Server secret exposed to the browser via public env prefix (VITE_SUPABASE_SERVICE_ROLE_KEY)

Variables prefixed VITE_ / NEXT_PUBLIC_ / REACT_APP_ are compiled into the JavaScript every visitor downloads. A secret here is public.

Fix: Rename without the public prefix, use it only in server code, and rotate it.

Evidence: .env:5 → VITE_SUPABASE_SERVICE_ROLE_KEY

CRITICAL Client code reads a secret from a public env var (VITE_SUPABASE_SERVICE_ROLE_KEY)

This value is inlined into the browser bundle at build time.

Fix: Use the anon/public key in client code; move privileged calls to a server route.

Evidence: src/lib/supabase.js:11

CRITICAL Supabase tables created without Row Level Security

Tables without RLS are readable/writable by anyone holding the public anon key.

Evidence: tables=messages, orders

HIGH AI endpoint /api/chat responds without authentication

An unauthenticated GET to /api/chat returned HTTP 200 and CORS allows any website (*). If this route forwards to an LLM, anyone can spend your credits or extract your system prompt. (Checked passively: no prompts were sent.)

Fix: Require a valid user session (e.g. supabase.auth.getUser(token)) before calling the model; restrict CORS to your own domain; add per-user rate limits and a spend cap at the LLM provider.

Evidence: GET /api/chat → 200; ACA0=*; referenced in JS

HIGH Source map publicly downloadable

/assets/index-Cntt6NGL.js.map is public and contains 52 original source files with full source code. Anyone can read your un-minified app code, including admin features and API logic.

Fix: Set build.sourcemap to false (Vite) / productionBrowserSourceMaps: false (Next), redeploy, and upload maps privately to your error tracker if needed.

Evidence: http://127.0.0.1:5180/assets/index-Cntt6NGL.js.map

HIGH Source map publicly downloadable

/assets/AdminDashboard-CwqJlJe5.js.map is public and contains 1 original source files with full source code. Anyone can read your un-minified app code, including admin features and API logic.

Fix: Set build.sourcemap to false (Vite) / productionBrowserSourceMaps: false (Next), redeploy, and upload maps privately to your error tracker if needed.

Evidence: http://127.0.0.1:5180/assets/AdminDashboard-CwqJlJe5.js.map

HIGH AI chat route has no authentication check

Anyone can call this route and spend your LLM credits or abuse your prompt/tools.

Fix: Verify the user's Supabase JWT before calling the model, and add per-user rate limits.

Evidence: server.js

HIGH Production source maps enabled in vite.config

build.sourcemap publishes your original source code next to the bundle.

Fix: Set build.sourcemap: false (or 'hidden' and upload maps privately).

Evidence: vite.config.js

MEDIUM Missing Content-Security-Policy

No Content-Security-Policy header was returned. CSP helps stop XSS and unauthorized script loads — common in vite-coded apps that pull in many third-party widgets.

Fix: Add a Content-Security-Policy header (start with default-src 'self'; script-src 'self'; and tighten from there). On Vercel/Netlify, set it in headers config; on Supabase-hosted static sites, configure your CDN.

MEDIUM Missing Strict-Transport-Security (HSTS)

HSTS was not set. Browsers may allow plaintext HTTP fallbacks or SSL stripping on first visit.

Fix: Serve only over HTTPS and add Strict-Transport-Security: max-age=31536000; includeSubDomains.

MEDIUM Web server sets no security headers

No CSP, HSTS, X-Frame-Options or nosniff headers are configured.

Fix: Set security headers on every response.

Evidence: `server.js`

LOW Missing X-Frame-Options

Without X-Frame-Options (or a CSP frame-ancestors directive), your app can be embedded in an attacker's iframe (clickjacking).

Fix: Add X-Frame-Options: DENY (or SAMEORIGIN), or set Content-Security-Policy: frame-ancestors 'none'.

LOW Missing X-Content-Type-Options

Browsers may MIME-sniff responses, which can turn a text file into executable script in older browsers.

Fix: Add X-Content-Type-Options: nosniff on all responses.

Checks passed

- `/.env` not exposed
- `/.git/config` not exposed
- `/.env.local` not exposed

Fix evidence: before and after

Screenshots captured by PreShip on the demonstration app. Row data and secrets are masked.

1. Supabase service_role key shipped to the browser

The admin key was compiled into the public JavaScript via a VITE_ variable. Fix: client uses the anon key only; the key was rotated, so the leaked one no longer works.

BEFORE · code · Supabase service_role key shipped to the browser	AFTER · code · fixed
<pre>— src/lib/supabase.js 8 // Admin client so the dashboard can see every order 9 export const supabaseAdmin = createClient(10 import.meta.env.VITE_SUPABASE_URL, 11 import.meta.env.VITE_SUPABASE_SERVICE_ROLE_KEY 12) — .env VITE_SUPABASE_SERVICE_ROLE_KEY=eyJh*****N_co</pre>	<pre>— src/lib/supabase.js 8 // Admin client so the dashboard can see every order 9 export const supabaseAdmin = createClient(10 import.meta.env.VITE_SUPABASE_URL, 11 import.meta.env.VITE_SUPABASE_ANON_KEY /* PreShip: service_role removed from client; privileged queries belong in a server route */ 12) — .env.example SUPABASE_SERVICE_ROLE_KEY=</pre>
BEFORE · live request · Supabase service_role key shipped to the browser	AFTER · live request · fixed
<pre>\$ curl -s \$SITE/assets/index-*.js grep -o 'eyJl-' # decode role claim /assets/index-CnttGNGl.js: JWT eyJh*****wPo role=anon /assets/index-CnttGNGl.js: JWT eyJh*****N_co role=service_role \$ curl \$SITE/rest/v1/orders -H 'apikey: eyJh*****N_co' # key copied from the bundle HTTP 200 3 rows returned (all customers' orders; values masked)</pre>	<pre>\$ curl -s \$SITE/assets/index-*.js grep -o 'eyJl-' # decode role claim /assets/index-Dw4-VSle.js: JWT eyJh*****XvGY role=anon → no service_role key in any browser bundle \$ curl \$SITE/rest/v1/orders -H 'apikey: eyJh*****N_co' # key copied from the bundle HTTP 401 {"code": "PGRS1301", "details": null, "hint": null, "message": "JWTError: JWTInvalidSignature"}</pre>

2. Row Level Security missing on orders and messages

Anyone with the public anon key could read every customer's orders and AI chat messages. Fix: RLS enabled with owner-only policies (migration 999).

BEFORE · code · Row Level Security missing on orders and messages	AFTER · code · fixed
<pre>— supabase/migrations/20260901120000_init.sql 13 create table public.orders (14 id bigint generated always as identity primary key, 15 user_id uuid references auth.users not null, 16 plan text not null, </pre>	<pre>— supabase/migrations/999_preshipsan_enable_rls.sql 6 alter table public.messages enable row level security; 7 create policy "messages_select_own" on public.messages for select to authenticated using ((select auth.uid() = user_id); 8 create policy "messages_insert_own" on public.messages for insert to authenticated with check ((select auth.uid() = user_id); 9 create policy "messages_update_own" on public.messages for update to authenticated using ((select auth.uid() = user_id) with check ((select auth.uid() = user_id); 10 create policy "messages_delete_own" on public.messages for delete to authenticated using ((select auth.uid() = user_id); 11 12 alter table public.orders enable row level security; 13 create policy "orders_select_own" on public.orders for select to authenticated using ((select auth.uid() = user_id); 14 create policy "orders_insert_own" on public.orders for insert to authenticated with check ((select auth.uid() = user_id); 15 create policy "orders_update_own" on public.orders for update to authenticated using ((select auth.uid() = user_id) with check ((select auth.uid() = user_id); 16 create policy "orders_delete_own" on public.orders for delete to authenticated using ((select auth.uid() = user_id); — supabase/migrations/20260901120000_init.sql 13 create table public.orders (14 id bigint generated always as identity primary key, 15 user_id uuid references auth.users not null, 16 plan text not null, </pre>
BEFORE · live request · Row Level Security missing on orders and messages	AFTER · live request · fixed
<pre>\$ curl \$SITE/rest/v1/orders -H 'apikey: <current public anon key>' # not logged in HTTP 200 3 rows readable by anyone [{"id": 1, "user_id": "****", "plan": "****", "amount_cents": "****", "delivery_address": "****", "status": "****", "created_at": "****"}, {"id": 2, "user_id": "****", "plan": "****", "amount_cents": "****", "delivery_address": "****", "status": "****", "created_at": "****"}] \$ curl \$SITE/rest/v1/messages -H 'apikey: <current public anon key>' # not logged in HTTP 200 2 rows readable by anyone [{"id": 1, "user_id": "****", "role": "****", "content": "****", "created_at": "****"}, {"id": 2, "user_id": "****", "role": "****", "content": "****", "created_at": "****"}]</pre>	<pre>\$ curl \$SITE/rest/v1/orders -H 'apikey: <current public anon key>' # not logged in HTTP 200 [] → 0 rows (RLS: anonymous users see nothing) \$ curl \$SITE/rest/v1/messages -H 'apikey: <current public anon key>' # not logged in HTTP 200 [] → 0 rows (RLS: anonymous users see nothing)</pre>

3. Unauthenticated /api/chat LLM proxy

Anyone could call the AI endpoint and spend the company's LLM credits. Fix: the server verifies a signed-in user's Supabase JWT and returns 401 otherwise.

BEFORE · code · Unauthenticated /api/chat LLM proxy	AFTER · code · fixed
<pre>— server.js 26 if (url.pathname.startsWith('/api/chat')) { 27 if (req.method === 'POST') { 28 const { messages } = JSON.parse(await readBody(req)) '{}'; 29 const reply = await callLLM(messages); 30 res.writeHead(200, { 'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*' }); 31 return res.end(JSON.stringify({ reply, model: 'gpt-4o-mini' })); 32 } 33 res.writeHead(200, { 'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*' }); 34 return res.end(JSON.stringify({ status: 'ok', model: 'gpt-4o-mini' })); 35 } 36 37 if (url.pathname.startsWith('/rest/v1')) {</pre>	<pre>— server.js 35 if (url.pathname.startsWith('/api/chat')) { 36 if (!verifyJwt(req)) { 37 res.writeHead(401, { 'Content-Type': 'application/json' }); 38 return res.end(JSON.stringify({ error: 'unauthorized' })); 39 } 40 if (req.method === 'POST') { 41 const { messages } = JSON.parse(await readBody(req)) '{}'; 42 const reply = await callLLM(messages); 43 res.writeHead(200, { 'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*' }); 44 return res.end(JSON.stringify({ reply, model: 'gpt-4o-mini' })); 45 } 46 res.writeHead(200, { 'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*' });</pre>
BEFORE · live request · Unauthenticated /api/chat LLM proxy	AFTER · live request · fixed
<pre>\$ curl -X POST \$SITE/api/chat -d '{"messages":[]}' # no login, no token HTTP 200 {"reply":"Try a quick veggie stir-fry! (stub reply to: \"What can I cook tonight?\"),\"model\":\"gpt-4o-mini\"}</pre>	<pre>\$ curl -X POST \$SITE/api/chat -d '{"messages":[]}' # no login, no token HTTP 401 {"error":"unauthorized"}</pre>

4. Missing security headers

No CSP, HSTS, X-Frame-Options or nosniff. Fix: headers set on every response by the server.

BEFORE · code · Missing security headers	AFTER · code · fixed
<pre>— server.js 23 const server = http.createServer(async (req, res) => { 24 const url = new URL(req.url, 'http://localhost'); 25 26 if (url.pathname.startsWith('/api/chat')) { 27 if (req.method === 'POST') { 28 const { messages } = JSON.parse(await readBody(req)) '{}'; 29 const reply = await callLLM(messages); 30 res.writeHead(200, { 'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*' }); 31 return res.end(JSON.stringify({ reply, model: 'gpt-4o-mini' })); 32 } 33 } 34 }</pre>	<pre>— server.js 24 const server = http.createServer(async (req, res) => { 25 // PreShip: security headers on every response 26 res.setHeader('Content-Security-Policy', "default-src 'self'; connect-src 'self'; img-src 'self' data:; style-src 'self' 'unsafe-inline'; frame-ancestors 'none'; base-uri 'self'"); 27 res.setHeader('Strict-Transport-Security', "max-age=63072000; includeSubDomains; preload"); 28 res.setHeader('X-Frame-Options', 'DENY'); 29 res.setHeader('X-Content-Type-Options', 'nosniff'); 30 res.setHeader('Referrer-Policy', 'strict-origin-when-cross-origin'); 31 res.setHeader('Permissions-Policy', 'camera=(), microphone=(), geolocation=()'); 32 33 const url = new URL(req.url, 'http://localhost');</pre>
BEFORE · live request · Missing security headers	AFTER · live request · fixed
<pre>\$ curl -sI \$SITE/ HTTP 200 content-security-policy: ✗ MISSING strict-transport-security: ✗ MISSING x-frame-options: ✗ MISSING x-content-type-options: ✗ MISSING referrer-policy: ✗ MISSING permissions-policy: ✗ MISSING</pre>	<pre>\$ curl -sI \$SITE/ HTTP 200 content-security-policy: default-src 'self'; connect-src 'self'; img-src 'self' data:; style-src 'self' 'unsafe-inline'; frame-ancestors 'none'; base-uri 'self' strict-transport-security: max-age=63072000; includeSubDomains; preload x-frame-options: DENY x-content-type-options: nosniff referrer-policy: strict-origin-when-cross-origin permissions-policy: camera=(), microphone=(), geolocation=()</pre>

5. Production source maps exposed

The full original source (including the admin dashboard and Supabase setup) was downloadable. Fix: build.sourcemap set to false.

BEFORE · code · Production source maps exposed	AFTER · code · fixed
<pre>— vite.config.js 5 export default defineConfig({ 6 plugins: [react()], 7 build: { 8 sourcemap: true, 9 }, 10 })</pre>	<pre>— vite.config.js 5 export default defineConfig({ 6 plugins: [react()], 7 build: { 8 sourcemap: false, 9 }, 10 })</pre>
BEFORE · live request · Production source maps exposed	AFTER · live request · fixed
<pre>\$ curl \$SITE/assets/index-CnttNGL.js.map HTTP 200 52 original source files exposed, e.g.: .../src/lib/supabase.js .../src/components/ChatAssistant.jsx .../src/App.jsx .../src/main.jsx sourcesContent (first lines): import { createClient } from '@supabase/supabase-js' export const supabase = createClient({</pre>	<pre>\$ curl \$SITE/assets/index-Dw4-VS1e.js.map HTTP 200 content-type: text/html → no source map (SPA fallback serves index.html) sourceMappingURL comment in bundle: absent</pre>

6. .env committed to git

Live keys were tracked in the repository. Fix: .env untracked and git-ignored, .env.example added, and every key in it rotated (it remains in old git history).

BEFORE · code · .env committed to git <pre>— .gitignore 1 node_modules 2 dist 3 *.local</pre>	AFTER · code · fixed <pre>— .gitignore 1 node_modules 2 dist 3 *.local 4 .env 5 .env.* 6 !.env.example</pre>
BEFORE · live request · .env committed to git <pre>\$ git ls-files grep .env tracked: .env \$ git show HEAD:.env # values masked VITE_SUPABASE_URL=http*****5180 VITE_SUPABASE_ANON_KEY=eyJh*****-wPo VITE_SUPABASE_SERVICE_ROLE_KEY=eyJh*****N_co OPENAI_API_KEY=sk-p*****DEMO POSTGREST_URL=http*****3001</pre>	AFTER · live request · fixed <pre>\$ git ls-files grep .env tracked: .env.example \$ cat .gitignore node_modules dist *.local .env .env.* !.env.example</pre>

Fixes

Key rotation (do this first)

Supabase

1. Supabase Dashboard > Project Settings > API: rotate the JWT secret / service_role key (this also rotates the anon key; update clients).
2. Never ship service_role to the browser; use it only in edge functions/servers.
3. Enable RLS on every table (see SQL above).

Openai

1. Go to platform.openai.com > API keys and revoke the exposed key.
2. Create a new key restricted to the project; set a monthly usage limit.
3. Store it only as a server env var (e.g. OPENAI_API_KEY in Vercel/Netlify settings), never with a VITE_/NEXT_PUBLIC_ prefix.
4. Review Usage for unexpected spend since the exposure date.

Supabase RLS policies (SQL)

```
-- PreShip: enable Row Level Security (RLS) on exposed tables.
-- Run in Supabase Dashboard > SQL Editor (test in a staging project first).
-- Assumes each row has an owner column `user_id` (uuid referencing auth.users).
-- Tables without an owner column get a deny-all-for-clients default; add policies deliberately.

alter table public.messages enable row level security;
create policy "messages_select_own" on public.messages for select to authenticated using ((select auth.uid()) = user_id);
create policy "messages_insert_own" on public.messages for insert to authenticated with check ((select auth.uid()) = user_id);
create policy "messages_update_own" on public.messages for update to authenticated using ((select auth.uid()) = user_id) with check ((select auth.uid()) = user_id);
create policy "messages_delete_own" on public.messages for delete to authenticated using ((select auth.uid()) = user_id);

alter table public.orders enable row level security;
create policy "orders_select_own" on public.orders for select to authenticated using ((select auth.uid()) = user_id);
create policy "orders_insert_own" on public.orders for insert to authenticated with check ((select auth.uid()) = user_id);
create policy "orders_update_own" on public.orders for update to authenticated using ((select auth.uid()) = user_id) with check ((select auth.uid()) = user_id);
create policy "orders_delete_own" on public.orders for delete to authenticated using ((select auth.uid()) = user_id);
```

Security headers: vercel.json

```
{
  "headers": [
    {
      "source": "/*.*",
      "headers": [
        {
          "key": "Content-Security-Policy",
          "value": "default-src 'self'; script-src 'self'; connect-src 'self' https://*.supabase.co; img-src 'self' data: http s;; style-src 'self' 'unsafe-inline'; frame-ancestors 'none'; base-uri 'self'"
        },
        {
          "key": "Strict-Transport-Security",
          "value": "max-age=63072000; includeSubDomains; preload"
        },
        {
          "key": "X-Frame-Options",
          "value": "DENY"
        }
      ]
    }
  ]
}
```

Security headers: Netlify_headers

```
/*
Content-Security-Policy: default-src 'self'; script-src 'self'; connect-src 'self' https://*.supabase.co; img-src 'self' data: https;; style-src 'self' 'unsafe-inline'; frame-ancestors 'none'; base-uri 'self'
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Referrer-Policy: strict-origin-when-cross-origin
Permissions-Policy: camera=(), microphone=(), geolocation=()
```

Security headers: next.config.js

```
/** @type {import('next').NextConfig} */
const securityHeaders = [
  { key: "Content-Security-Policy", value: "default-src 'self'; script-src 'self'; connect-src 'self' https://*.supabase.co; img-src 'self' data: https:; style-src 'self' 'unsafe-inline'; frame-ancestors 'none'; base-uri 'self'" },
  { key: "Strict-Transport-Security", value: "max-age=63072000; includeSubDomains; preload" },
  { key: "X-Frame-Options", value: "DENY" },
  { key: "X-Content-Type-Options", value: "nosniff" },
  { key: "Referrer-Policy", value: "strict-origin-when-cross-origin" },
  { key: "Permissions-Policy", value: "camera=(), microphone=(), geolocation=()" }
];

const nextConfig = {
  reactStrictMode: true,
  productionBrowserSourceMaps: false,
  async headers() {
    return [{ source: "/(.*)", headers: securityHeaders }];
  },
};

module.exports = nextConfig;
```

Pull-request-ready patch (git diff)

Display copy with secrets redacted. Download the applicable patch from (available 7 days).

```
PreShip launch fixes. Apply from the repo root:
git rm --cached .env # stop tracking (keep your local copy, then rotate its keys)
git apply preshipscan.patch

diff --git a/.env.example b/.env.example
new file mode 100644
index 00000000000000000000000000000000..c547aa5d3b78d0d767883c63b42b8c0e9d5e3faa
--- /dev/null
+++ b/.env.example
```

```

@@ -0,0 +1,5 @@
+VITE_SUPABASE_URL=
+VITE_SUPABASE_ANON_KEY=
+SUPABASE_SERVICE_ROLE_KEY=
+OPENAI_API_KEY=
+POSTGREST_URL=
diff --git a/.gitignore b/.gitignore
index 804bd4a4aec2df8df75899c8958103151b32b608..20e03f61dd3f1c317b38245ccc5ca01ed68c6fc0 100644
--- a/.gitignore
+++ b/.gitignore
@@ -1,3 +1,6 @@
 node_modules
 dist
 *.local
+.env
+.env.*
+!.env.example
diff --git a/lib/verify-jwt.js b/lib/verify-jwt.js
new file mode 100644
index 0000000000000000000000000000000000000000..86d18e2b603d10627301fc5e9f25d22b44830838
--- /dev/null
+++ b/lib/verify-jwt.js
@@ -0,0 +1,13 @@
+// Added by PreShip: verify a Supabase access token (HS256) server-side.
+import crypto from 'node:crypto'
+
+export function verifyJwt(req) {
+  const token = (req.headers.authorization || '').replace(/^Bearer /, '')
+  const [h, p, s] = token.split('.')
+  if (!h || !p || !s || !process.env.SUPABASE_JWT_SECRET) return null
+  const expected = crypto.createHmac('sha256', process.env.SUPABASE_JWT_SECRET).update(`${h}.${p}`).digest('base64url')
+  if (expected.length !== s.length || !crypto.timingSafeEqual(Buffer.from(expected), Buffer.from(s))) return null
+  const claims = JSON.parse(Buffer.from(p, 'base64url').toString())
+  if (claims.role !== 'authenticated' || !claims.sub || (claims.exp && claims.exp * 1000 < Date.now())) return null
+  return claims
+}
diff --git a/server.js b/server.js
index a5fa5213fb7cfd393366caceff4d4a4ef385070b..d2447b5b3cb88bebf5a5d645f181d8ae6b731c7 100644
--- a/server.js
+++ b/server.js
@@ -2,6 +2,7 @@
import http from 'node:http'
import fs from 'node:fs'
import path from 'node:path'
+import { verifyJwt } from './lib/verify-jwt.js'

const PORT = process.env.PORT || 5180
const DIST = path.resolve('dist')
@@ -21,9 +22,21 @@ async function readBody(req) {
}

const server = http.createServer(async (req, res) => {
+ // PreShip: security headers on every response
+ res.setHeader('Content-Security-Policy', "default-src 'self'; connect-src 'self'; img-src 'self' data:; style-src 'self' 'unsafe-inline'; frame-ancestors 'none'; base-uri 'self'")
+ res.setHeader('Strict-Transport-Security', 'max-age=63072000; includeSubDomains; preload')
+ res.setHeader('X-Frame-Options', 'DENY')
+ res.setHeader('X-Content-Type-Options', 'nosniff')
+ res.setHeader('Referrer-Policy', 'strict-origin-when-cross-origin')
+ res.setHeader('Permissions-Policy', 'camera=(), microphone=(), geolocation=()')
+
  const url = new URL(req.url, 'http://localhost')

  if (url.pathname.startsWith('/api/chat')) {
+   if (!verifyJwt(req)) {
+     res.writeHead(401, { 'Content-Type': 'application/json' })
+     return res.end(JSON.stringify({ error: 'unauthorized' }))
+   }

```

```

    if (req.method === 'POST') {
      const { messages } = JSON.parse(await readBody(req)) || '{}';
      const reply = await callLLM(messages)
    }
  }
}

diff --git a/src/lib/supabase.js b/src/lib/supabase.js
index a01a464b0446a6f67591ff8f2a6376fd96ee2fa0..9200a98a293421fc6ba209304ddf1fcf24a7ca38 100644
--- a/src/lib/supabase.js
+++ b/src/lib/supabase.js
@@ -8,5 +8,5 @@ export const supabase = createClient(
  // Admin client so the dashboard can see every order
  export const supabaseAdmin = createClient(
    import.meta.env.VITE_SUPABASE_URL,
    import.meta.env.VITE_SUPABASE_SERVICE_ROLE_KEY
  + import.meta.env.VITE_SUPABASE_ANON_KEY /* PreShip: service_role removed from client; privileged queries belong in a server route */
  )
}

diff --git a/supabase/migrations/999_preshipscan_enable_rls.sql b/supabase/migrations/999_preshipscan_enable_rls.sql
new file mode 100644
index 0000000000000000000000000000000000000000..372fe85ce09fbf2c99b3dedbca700d164505416f
--- /dev/null
+++ b/supabase/migrations/999_preshipscan_enable_rls.sql
@@ -0,0 +1,17 @@
+-- PreShip: enable Row Level Security (RLS) on exposed tables.
+-- Run in Supabase Dashboard > SQL Editor (test in a staging project first).
+-- Assumes each row has an owner column `user_id` (uuid referencing auth.users).
+-- Tables without an owner column get a deny-all-for-clients default; add policies deliberately.
+
+alter table public.messages enable row level security;
+create policy "messages_select_own" on public.messages for select to authenticated using ((select auth.uid()) = user_id);
+create policy "messages_insert_own" on public.messages for insert to authenticated with check ((select auth.uid()) = user_id);
+create policy "messages_update_own" on public.messages for update to authenticated using ((select auth.uid()) = user_id) with check ((select auth.uid()) = user_id);
+create policy "messages_delete_own" on public.messages for delete to authenticated using ((select auth.uid()) = user_id);
+
+alter table public.orders enable row level security;
+create policy "orders_select_own" on public.orders for select to authenticated using ((select auth.uid()) = user_id);
+create policy "orders_insert_own" on public.orders for insert to authenticated with check ((select auth.uid()) = user_id);
+create policy "orders_update_own" on public.orders for update to authenticated using ((select auth.uid()) = user_id) with check ((select auth.uid()) = user_id);
+create policy "orders_delete_own" on public.orders for delete to authenticated using ((select auth.uid()) = user_id);
+
diff --git a/vite.config.js b/vite.config.js
index a6ab9e94c4284e33e4f8a29b64103af84c4563d0..30a3acfd5284775e596fa8633f5d04283402230d 100644
--- a/vite.config.js
+++ b/vite.config.js
@@ -5,6 +5,6 @@ import react from '@vitejs/plugin-react'
export default defineConfig({
  plugins: [react()],
  build: {
    sourcemap: true,
    sourcemap: false,
  },
})

```

Disclaimer. IMPORTANT DISCLAIMER (DRAFT - requires attorney review). This report is an automated, point-in-time security review of publicly accessible resources and, where supplied, source code, performed at the request of a party who represented that they own or are authorized to test the target. It is NOT a penetration test, NOT a security audit or certification, and NOT evidence of compliance with any standard or law (including SOC 2, ISO 27001, HIPAA, PCI DSS, or GDPR). It is not a guarantee that the application is secure or without vulnerabilities; automated checks can miss issues and can report false positives. Findings and fixes are provided AS IS, without warranties of any kind, express or implied. Suggested fixes must be reviewed and tested by the customer before deployment. Secrets detected during scanning are redacted and are not stored. To the maximum extent permitted by law, total liability of the provider for any claim related to this report is limited to the fees paid for it. The customer is responsible for having authorization to scan the target and agrees to indemnify the provider for any unauthorized scan.