Healio.AI — Senior Developer Critical Analysis

> ***Reviewed:*** *Every directory and ~30 key files across frontend, backend, diagnosis engine, auth, security, and project hygiene.*

🔴 SEVERITY: CRITICAL

1. Leaked API Key in Committed `.env` File

backend/.env contains a **live Resend API key** (`re_NrDRKumJ_...`) committed to Git. This is a **P0 security incident**.

> *[!CAUTION]* ***Action Required:*** *Immediately rotate this key in Resend dashboard, add `backend/.env` to `.gitignore` , and use `git filter-branch` or BFG Repo Cleaner to purge history.*

---

2. `getNotifications()` Bug — `unreadOnly` Filter Silently Ignored

In api.ts:383-405, the function builds a filtered query chain (`query`) but **never executes it**. Instead, it creates a brand-new unfiltered query at line 394. Every call to `getNotifications(userId, true)` returns **all** notifications, not just unread ones.

```
async getNotifications(userId: string, unreadOnly = false) {
    let query = supabase
        .from('notifications')
        .select('*')
        .eq('user_id', userId)
        .order('created_at', { ascending: false });

    if (unreadOnly) {
        query = query.eq('is_read', false);
    }

-    const { data, error } = await supabase
-        .from('notifications')
-        .select('*')
-        .eq('user_id', userId)
-        .order('created_at', { ascending: false });
+    const { data, error } = await query;
```

---

3. `createAppointment()` Hardcodes Consultation Fee

In api.ts:277, `consultation_fee: 500` is hardcoded rather than pulled from the doctor's profile. This will **silently overcharge or undercharge** every patient.

---

4. No Server-Side Route Protection

rbac.ts provides role-checking utilities, but:

- middleware.ts only refreshes sessions — it performs **zero role-based access control**.
- A patient can directly navigate to `/admin` or `/doctor` routes by typing the URL.
- All API routes lack server-side auth verification.

> *[!WARNING] The RBAC system is cosmetic. Any authenticated user can access any portal. The middleware must enforce `isRouteAllowed()` checks after `getUser()`.*

🟠 SEVERITY: HIGH — Diagnosis Engine Architecture

5. "Bayesian" Engine Is Not Actually Bayesian

The core engine in engine.ts claims to use Bayesian inference but implements a **manual log-probability accumulation** with hardcoded boost values (e.g., `+3.0`, `+2.0`, `-5.0`) passed through a sigmoid. This is not Bayesian – it's a **weighted scoring function with a squashing activation**.

What a proper Bayesian network would provide:

Feature	Current Implementation	Proper Bayesian Network / MCMC
Priors	Flat map of 5 prevalence buckets	Learned priors from epidemiological data
Likelihood	Hardcoded boosts (`+3.0`)	`P(symptom
Posterior	Sigmoid of summed log-scores	True `P(disease
Uncertainty	Heuristic width heuristics	Posterior distribution sampling (MCMC/HMC)
Conditional independence	None — all symptoms treated independently	Directed acyclic graph encoding dependencies
Calibration	None	Platt scaling or isotonic regression
Evidence updates	Re-score from scratch	Incremental belief propagation

> *[!IMPORTANT] **Recommendation:** Replace the scoring engine with a proper **Bayesian Network** using libraries like `pgmpy` (Python) or `bayes-server`, with:*
>
> - *A DAG encoding symptom-disease-comorbidity relationships*
> - *MCMC sampling (Metropolis-Hastings or NUTS) for posterior estimation*
> - *Calibrated confidence intervals from posterior distributions*
> - *Network structure learned from clinical data (ICD-10 mappings)*

---

6. Duplicate Scoring Functions

engine.ts contains **both** `calculateScore()` (lines 228-455) AND `calculateBayesianScore()` (lines 463-731). The former is a dead function — never called. 1,200 lines of engine code that should be ~400.

---

7. Symptom Matching is Fragile String Matching

All symptom detection relies on `string.includes()` against free-text notes. This produces:

- **False positives:** "I do NOT have chest pain" → matches "chest" + "pain"
- **Missed detections:** "hurts in my torso" won't match "chest"
- The negation regex (`/no|not|without/`) is too simplistic — fails on "I have nothing but chest pain"

Suggestion: Use NLP tokenization + NER (Medical NER like SciSpacy or BioBERT), or at minimum build a proper token-level matching system instead of substring searches.

---

8. Mock Embedding Fallback Returns Random Vectors

In embeddings/route.ts, when no OpenAI key is available, the API returns a **random 1536-dim vector**. This means vector search in retrieval.ts returns **random conditions** — the entire diagnosis pipeline degrades to the local keyword fallback silently, with no indication to the user.

---

9. Retrieval Falls Back to Returning ALL Conditions

In retrieval.ts:70, when vector search yields < 5 results, the system returns `Object.values(CONDITIONS)` — the **entire conditions catalog** — bypassing semantic relevance entirely. This turns the "smart" retrieval into brute force.

🟡 SEVERITY: MEDIUM — Code Quality & Architecture

10. Middleware Uses Deprecated Supabase Cookie Pattern

middleware.ts uses `get/set/remove` individual cookie methods. The modern `@supabase/ssr` library expects `getAll/setAll` (used correctly in supabaseServer.ts but not in middleware).

---

11. 277-Line Mock Data File in Production `lib/`

mockData.ts contains 8 hardcoded patient records with PII-like data (names, emails, phone numbers, medical conditions). This is imported by production components and should:

- Be moved to `__tests__/fixtures/` or `data/seed/`
- Never ship in production bundles

---

12. Entire Legacy Vite App Still in Repo

A complete legacy/ directory contains a full Vite application with its own `node_modules/`, `package.json`, and `src/`. This adds **30+ files** and bloats the repo by ~30MB+.

---

13. Project Root is a Dumping Ground

The root directory contains **20+ loose files** that don't belong:

File Pattern	Count	Action
`lint_*.txt`, `lint_*.log`, `lint_*.json`	8	Delete or add to `.gitignore`
`tsc_*.txt`, `tsc_*.log`	5	Delete
`test_*.txt`, `test_*.sql`	6	Move to `tests/` or `scripts/`
`build_*.txt`, `build_*.log`	2	Delete
`*.sql` (loose migrations)	6	Move to `supabase/migrations/`
`app_errors.txt`, `diagnosis_errors.txt`	3	Delete
`verification_result.txt`	1	Delete

---

14. Python Backend Has No Authentication

backend/main.py has rate limiting but **zero authentication**. Any person on the internet can:

- Upload files to the server
- Send emails through the `/api/email/*` endpoints (spam vector)
- The email endpoints don't verify the requesting user owns the target email

---

15. Duplicate Dependency in `requirements.txt`

requirements.txt lists `python-multipart` twice (lines 6 and 8). Dependencies are also unpinned (no version for `fastapi`, `uvicorn`).

---

16. `react-simple-maps` Still Installed Alongside `react-google-charts`

package.json still lists `react-simple-maps@3.0.0` even though it was previously replaced with `react-google-charts`. This adds ~150KB of dead weight to the bundle.

---

17. Typo in Backend Response

In main.py:302:

```
return {" status": "success", ...}  # Note the leading space in " status"
```

This will cause client-side parsing issues if checking `response.status`.

---

18. Supabase Client Uses Non-Null Assertion + Fallback Chain

In supabase.ts:

```
const supabaseAnonKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
    || process.env.NEXT_PUBLIC_SUPABASE_PUBLISHABLE_DEFAULT_KEY!;
```

The `!` (non-null assertion) makes the `||` fallback dead code — TypeScript tells the compiler the first value is always truthy. Use conditional checks instead.

---

19. `api.ts` Makes Excessive Supabase Round-Trips

Functions like getPatientAppointments() make **3 sequential Supabase queries** (appointments → doctors → profiles) that could be a single query with JOINs:

```
SELECT a.*, d.specialty, p.full_name
FROM appointments a
JOIN doctors d ON d.user_id = a.doctor_id
JOIN profiles p ON p.id = d.user_id
WHERE a.patient_id = $1
```

---

20. No Error Boundaries in React Components

Large page components (20K+ line dashboard files) have no React Error Boundaries. A single crashed API call takes down the entire dashboard.

🔵 SEVERITY: IMPROVEMENT — Strategic Suggestions

21. Replace Client-Side Diagnosis Engine with Server-Side Service

The entire 1,158-line diagnosis engine runs **in the browser**. This means:

- Every user downloads 54KB of diagnostic logic
- No ability to update the model without a frontend deploy
- No audit trail of diagnoses for compliance
- Easy to reverse-engineer the scoring

Recommendation: Move to a server-side microservice (Python/FastAPI) with:

- A proper Bayesian Network engine (pgmpy, PyMC)
- MCMC posterior sampling for calibrated confidence
- Database-driven condition rules (updatable without deploys)
- Audit logging for every diagnosis

---

22. Add a Proper Observability Stack

The codebase uses `console.log/error` everywhere with no:

- Structured logging (e.g., Pino, Winston)
- Error tracking (e.g., Sentry)
- Performance monitoring (e.g., OpenTelemetry)
- Audit trail for medical decisions

---

23. Implement Proper TypeScript Strictness

Several files use `any` types freely:

- `api.ts` line 44: `data: any`
- `api.ts` line 13: `availability: any`
- `retrieval.ts` line 57: `as any[]`
- Multiple `.map((a: any) => ...)` casts

Enable `"strict": true` and `"noImplicitAny": true` in `tsconfig.json`.

---

24. Implement Database-Driven Condition Management

Currently, conditions are either:

- Hardcoded in TypeScript files (19 files in `conditions/`)
- Or fetched via vector search

A proper CMS or admin panel for managing conditions would allow medical professionals to update diagnostic rules without engineering involvement.

---

25. Add API Rate Limiting on Frontend Routes

The Next.js API routes (`/api/embeddings` , `/api/admin/*` , `/api/analytics`) have **no rate limiting** — only the Python backend does. These routes are directly exploitable.

---

26. Implement Proper Testing

The project has:

- One Playwright config but minimal E2E tests
- Test files in the diagnosis engine (`test_clinical_engine.ts`) but no test runner or CI integration
- No unit tests for `api.ts` , RBAC, scheduling, or any component

---

27. `package.json` Still Named `temp-app`

The package name is `"temp-app"` — likely from initial scaffolding. Should be `"healio-ai"` .

Summary Priority Matrix

Priority	Issue	Impact	Effort
🔴 P0	Leaked API key in `.env`	Security breach	30 min
🔴 P0	No server-side route protection	Any user = admin	2-4 hours
🔴 P0	`getNotifications` bug	Silent data issue	5 min fix
🔴 P1	Backend has no auth	Open email spam	2-3 hours
🟠 P1	Replace pseudo-Bayesian engine	Core product quality	2-4 weeks
🟠 P2	Remove dead `calculateScore()` function	Code cleanliness	15 min
🟡 P2	Fix embedding fallback to return error, not random vectors	Diagnosis accuracy	1 hour
🟡 P2	Remove mock data from production	Bundle size, data hygiene	30 min
🟡 P3	Delete legacy dir + root dump files	Repo hygiene	1 hour
🟡 P3	Upgrade middleware cookie pattern	Auth reliability	30 min
🔵 P3	Move engine server-side	Architecture	1-2 weeks
🔵 P3	Add observability + testing	Reliability	Ongoing