HIPAA & Security Audit Report

Client	US Private Practice
Date	Aug 7, 2026
Prepared by	Roman Zaporozhets, Founder & CEO

Contents

01. Summary 3

02. What We Reviewed 3

03. Findings Overview 4

04. Must Fix Before Launch 4

1. Database security rules are wide open
2. BAAs are missing with some vendors
3. AI notes feature sends patient data insecurely
4. Audit logs can be edited/changed
5. Audit trail has gaps
6. No database backups
7. No MFA
8. HIPAA-compliance documentation

05. Should Fix Before or Shortly After Launch 8

9. Public intake form is unprotected
10. Password reset lets admin see the password
11. No error monitoring
12. No file size limits on uploads
13. Admin session stays active forever
14. Weak password requirements
15. No staging environment
16. Deactivated users can still access briefly
17. Patient data in system logs

06. Code Review 11

07. UI/UX Review 12

08. Load / Scalability Assessment 13

09. Remediation Plan 15

01. Summary

Good foundation, but it has critical HIPAA & Security gaps that need to be addressed before it can go to production.

We went through the entire system - admin dashboard, iPad app, cloud backend. Reviewed the code, ran everything locally, mapped out how patient data moves through the system.

The app works and the core decisions are right. Authentication, role-based access, audit logging, encryption - all there. The developer built this with healthcare in mind, and it shows.

But there are real gaps that need to be fixed before you can use this with actual patient data. The biggest one - the database security rules are completely open. All the access controls in the app code can be bypassed. There are two more critical ones - how the AI feature handles patient data, and missing BAAs with vendors that receive it.

Beyond security - the code needs cleanup over time, the design is inconsistent but fine for launch, and the app slows down a lot as the number of clients grows.

Findings by severity

3	5	8	1
Critical	High	Medium	Low

02. What We Reviewed

- **Backend (Cloud Functions)** - all API endpoints, database rules, security, audit logging
- **Admin dashboard** - session management, data handling, security
- **iOS app** - therapist mode, kiosk mode, OpenAI integration
- **Infrastructure** - encryption, backups, monitoring, deployment
- **HIPAA compliance** - data flow, access controls, audit trail, BAA requirements

This is a point-in-time review of the code and configuration as of the audit date. It's not a penetration test or a HIPAA certification.

03. Findings Overview

#	Finding	Severity	Status
1	Database security rules are wide open	Critical	Fixed
2	BAAs are missing with some vendors	Critical	Signed
3	AI notes feature sends patient data insecurely	Critical	Fixed
4	Audit logs can be edited/changed	High	Fixed
5	Audit trail has gaps	High	Fixed
6	No database backups	High	Fixed
7	No MFA	High	Fixed
8	HIPAA-compliance documentation	High	Fixed
9	Public intake form is unprotected	Medium	Fixed
10	Password reset lets admin see the password	Medium	Fixed
11	No error monitoring	Medium	Fixed
12	No file size limits on uploads	Medium	Fixed
13	Admin session stays active forever	Medium	Fixed
14	Weak password requirements	Medium	Fixed
15	No staging environment	Medium	Fixed
16	Deactivated users can still access briefly	Medium	Fixed
17	Patient data in system logs	Low	Fixed

Findings describe the system at the time of the audit. Status shows the result after remediation.

04. Must Fix Before Launch

These need to be resolved before any real patient data goes into the system. The one exception was MFA - agreed with the client to do right after launch.

1. Database security rules are wide open

Critical

Right now the database allows anyone with basic technical knowledge to read, write, and delete all patient data - bypassing everything the app does to protect it. This is the #1 issue.

The app has proper access controls in its code, but the database itself doesn't enforce any of them. All someone needs is the Firebase project ID, which isn't hard to get. The real risk isn't a targeted attack, but the bots that scan the internet.

> **Fix:** Replace the default rules with proper role-based rules that check who's making the request and what they're allowed to do.

Where	Firestore security rules
HIPAA	Access control, §164.312(a)(1)
Status	**Fixed, checked**

2. BAAs are missing with some vendors

Critical

Patient data goes to these vendors, but there's no BAA in place:

- **Google Cloud** - needs to be accepted in the Google Cloud console
- **OpenAI** - needs a Data Processing Addendum/BAA signed for the API account

> **Fix:** Accept the Google Cloud BAA and sign the OpenAI BAA before any real patient data goes in.

HIPAA	Business associate contracts, §164.308(b)(1)
Status	**Signed**

3. AI notes feature sends patient data insecurely

Critical

The iOS app sends patient data directly to OpenAI for AI note generation. Two problems here:

1. The OpenAI key is stored on the device. Someone with technical knowledge could extract it from the app.
2. Patient names, date of birth, and email are included in what gets sent to OpenAI. OpenAI only needs the clinical details to generate notes - it doesn't need to know who the patient is.

Fix: Route AI requests through the cloud backend instead of directly from the iOS app. Store the key on the server. Strip patient names and contact info before sending - the AI only needs treatment details to write notes.

Where	iOS app, AI notes
HIPAA	Minimum necessary, §164.502(b) · Transmission security, §164.312(e)(1)
Status	**Fixed, checked**

---

4. Audit logs can be edited/changed

High

Because of the open database rules (#1), anyone could edit or delete audit log entries.

Fix: When the database rules are fixed, set audit logs to write-only. Once created, nobody can modify or delete them.

Where	Firestore security rules, audit log collection
HIPAA	Audit controls, §164.312(b) · Integrity, §164.312(c)(1)
Status	**Fixed, checked**

---

5. Audit trail has gaps

High

HIPAA requires logging every time someone accesses patient data. The system logs when data is created, updated, or deleted - but not when someone views records. If a staff member looked at records they shouldn't have, there'd be no evidence.

Also - staff SSNs can show up in audit logs when profiles are updated. Sensitive values should never be in logs. Log what was accessed, not the actual data.

Fix: Add logging for all data viewing. Remove sensitive values from log entries.

HIPAA	Audit controls, §164.312(b)
Status	**Fixed, checked**

6. No database backups

High

No automated backups configured. If data gets accidentally deleted or corrupted, there's no way to recover it. HIPAA requires maintaining retrievable copies of patient data.

> **Fix:** Turn on automated daily backups in the Firebase console.

HIPAA	Data backup plan, §164.308(a)(7)(ii)(A)
Status	**Fixed, checked**

7. No MFA

High

Staff log in with just email and password - no second factor like a code from an authenticator app. Not required by HIPAA yet, but strongly recommended. Especially for admin accounts that can access all patient data.

> **Fix:** Enable MFA in Firebase Identity Platform and implement it on the code side - enrollment, the second step at login, and recovery if someone loses their phone.

HIPAA	Person or entity authentication, §164.312(d) · required in the proposed Security Rule update
Status	**Fixed, checked**

8. HIPAA-compliance documentation

High

HIPAA requires these documents:

- **Security Risk Analysis (SRA)** - where patient data is, what could go wrong, how it's protected
- **Breach Notification Procedure** - who needs to be notified and when, if patient data is breached
- **Incident Response Plan** - what to do when something goes wrong
- **Information Security Policies** - how systems and data are protected
- **Emergency Access Procedure** - how staff get to patient data in an emergency
- **Disaster Recovery Plan** - how systems and data are restored after an outage or data loss
- **Data Retention & Disposal Policy** - how long data is kept, how it's destroyed
- **Employee Training Policy** - HIPAA training requirements, proof of completion

Fix: Put the required policies in place.

HIPAA	Policies and documentation, §164.316
Status	**Fixed, checked**

05. Should Fix Before or Shortly After Launch

Not blocking launch, but should be addressed soon.

---

9. Public intake form is unprotected

Medium

The public form feature (patients fill out intake forms via a link) has no protection:

- No limit on submissions - someone could flood the system with thousands of fake records
- Any website can submit data to this endpoint

Fix: Add rate limiting (5-10 submissions per hour per visitor) and restrict which websites can submit forms.

Where	Public form endpoint
Status	**Fixed, checked**

---

10. Password reset lets admin see the password

Medium

When an admin resets someone's password, they set the new password directly - so the admin knows it. It should send a reset link to the staff member's email instead. The admin never sees the new password.

Fix: Use Firebase's built-in password reset email instead of admin-set passwords.

HIPAA	Person or entity authentication, §164.312(d)
Status	**Fixed, checked**

11. No error monitoring

Medium

No system to detect and alert when errors happen in production. If something breaks, nobody knows until someone complains.

> **Fix:** Set up Google Cloud error reporting and alerts, so the team gets notified when errors occur. It's already covered by the Google Cloud BAA - no need for a third-party tool.

HIPAA	Information system activity review, §164.308(a)(1)(ii)(D)
Status	**Fixed, checked**

12. No file size limits on uploads

Medium

Image and document uploads accept files of any size. Someone could upload massive files and eat up all the storage.

> **Fix:** Add a size check - reject anything over 10MB.

Status	**Fixed, checked**

13. Admin session stays active forever

Medium

The admin dashboard stays logged in even after closing the browser. No inactivity timeout either. If a laptop gets lost or left unattended, someone could open the browser and have full access to everything.

> **Fix:** End the session when the browser closes. Add an inactivity timeout.

HIPAA	Automatic logoff, §164.312(a)(2)(iii)
Status	**Fixed, checked**

14. Weak password requirements

Medium

Passwords only need 6 characters, no complexity requirements. Someone could use "123456."

Fix: Minimum 8 characters, require a mix of letters and numbers.

HIPAA	Password management, §164.308(a)(5)(ii)(D)
Status	**Fixed, checked**

---

15. No staging environment

Medium

Only one environment - production. Any changes or testing happen against the live system.

Fix: Set up a proper, separate staging environment for development.

Status	**Fixed, checked**

---

16. Deactivated users can still access briefly

Medium

When an admin deactivates someone's account, they can keep using the system for up to an hour with their existing session.

Fix: Implement immediate token revocation when a user is deactivated.

HIPAA	Termination procedures, §164.308(a)(3)(ii)(C)
Status	**Fixed, checked**

---

17. Patient data in system logs

Low

The backend logs patient emails and other identifiers to Google Cloud Logging. Not publicly accessible, but visible to anyone with project access and may be retained indefinitely.

Fix: Remove or redact patient identifiers from log messages.

Status	**Fixed, checked**

06. Code Review

Summary

The codebase does the job. Some parts look good, and it works. But it was clearly built under pressure - ship fast, deal with it later. And "later" never came.

The pattern is the same across all three codebases: the first thing gets built properly, then everything after it is copy-pasted from that first thing. Nobody went back to clean up. Files grew to 2,000+ lines, the same logic lives in 3-4 places, and each part of the app handles the same thing slightly differently.

Some modules (like Auth) were built properly, but then we see all records loaded into memory and no pagination at all, which hangs the app - that's a junior mistake.

Admin dashboard (Flutter)

The codebase is not bad for an MVP. Core decisions are right - Firebase as the API layer, role-based access, audit logging built in. The code does the job and the app works, but there are many shortcuts and maintainability concerns.

The main concern is a poorly chosen architecture pattern. It's called GetX / MVC. It lets you ship fast, but it also lets you do anything and doesn't enforce structure. That leads to spaghetti code and makes it difficult to maintain over time. And that's what we see here:

- **Files are too big** - several are 1,500-2,000+ lines doing too many things at once
- **Code duplication everywhere** - auth checks, error handling, token retrieval, HTTP calls all copied across files instead of shared
- **Inconsistent patterns** - the same things handled differently in different parts of the codebase
- **No reusable UI components** - every page builds its own UI from scratch
- **Few tests on critical paths** - only about a third of the backend functions that handle patient data have tests
- **No error handling strategy** - the admin panel shows raw exception strings like "HTTP Error: 500" to clinic staff. No friendly messages, no retry logic

Not urgent, but worth cleaning up over time. It makes every future change safer and faster.

Backend (Cloud Functions)

The backend code works and handles the basic stuff - authentication, RBAC, audit logging. But it was built fast, and it shows. The same logic is copied across multiple files instead of shared, which means fixing something in one place doesn't fix it everywhere.

It's ok for launch, but needs cleanup before adding new features or scaling up.

iOS app

Some parts of the iOS app are well done. But then the app grew, code was copied instead of shared, and files got way too big. The architecture was started right but wasn't followed through - only 2 out of 15+ screens actually use it.

07. UI/UX Review

Summary – kiosk / therapist app

The design is functional but inconsistent. Buttons, input fields, dialogs, spacing and typography vary from screen to screen. There's no shared design system - each page was styled independently, which is why things look slightly different everywhere.

It's not broken - patients and therapists can use it without issues, but it doesn't feel like one cohesive product.

For launch, this is fine. Down the road it needs to be fixed to make it consistent and professional looking.

Button inconsistency

- "Start" button on the kiosk welcome screen is a rounded pill shape
- "Next →" on service selection is a different rounded pill
- "Submit Form" at the bottom is a full-width bar with different rounding
- "Submit" in the confirmation modal is a smaller rounded button - yet another style
- "Done" on the thank you screen is back to the pill shape, but a different size
- "Start →" on the therapist forms list is yet another shape

That's 5-6 different button styles across the app. Should be 2-3 max (primary, secondary, destructive).

Native vs custom mixing

- Cancel confirmation uses a custom modal (white card with rounded corners)
- Submit confirmation also uses a custom modal, but with different styling (has an icon, different button layout)
- Some errors use native iOS alerts
- Some use custom overlays

Spacing

- Form fields have inconsistent vertical gaps between them
- The "Select Service" screen has too much empty space above and below the single option
- "Find Your Profile" has the email field oddly narrow compared to the full-width fields on the form screen

Field styling

- Kiosk form fields have a light border with rounded corners
- Therapist form fields (Objective, Assessment, Treatment Plan) have a slightly different border style and more padding
- These should look the same - same app, same field component

Typography

- "Select Service" uses one heading weight
- "Find Your Profile" uses a different one
- "Massage Forms" on the therapist screen is bold sans-serif, while kiosk headings use serif

08. Load / Scalability Assessment

Summary

The app works well at small volume but doesn't scale. There's no pagination anywhere - every time you open the Clients page, it loads every single client in the database. Same for the dashboard and the therapist's form queue on the iPad.

At 500 clients, the dashboard takes 3-4 seconds. At 2,000, the admin panel becomes unusable - 10+ second load times and the browser freezes when scrolling. The therapist's iPad also slows down

significantly. How fast this becomes a problem depends on how many patients come through the door.

The backend loads too much data, and the frontend renders all of it at once. The fix is adding pagination (load 50 at a time instead of everything) and storing client names directly on form records, so the system doesn't have to look them up separately every time.

Performance issues

1. **No pagination anywhere except Audit Logs.** If you have 1,000 clients in the database, the Clients page loads all 1,000. Same for Forms and everything else. With 2,000+ clients it just hangs the browser - the page takes about 10 seconds to load and it's laggy when scrolling. Unusable.
2. **Every form fetch also fetches the client.** To get the client's name, it fetches the client first. So if you request 50 forms, it also fetches 50 clients, which doubles the time.
3. **listClients runs 2 extra queries** to get the form count. Inefficient queries that could be avoided.
4. **getFormsBySubcategory loads ALL forms across ALL clients into memory,** then filters by subcategory. This is the therapist's main work queue - the screen they see when they select "Massage" to see today's patients. At 6,000+ forms, it loads everything just to show the 20 massage forms from today.
5. **getAuditStats loads 30 days of ALL audit logs into memory to count them.** Instead of using Firestore aggregation queries (count() with filters), it downloads every log document and counts in JavaScript. At high audit volume this will time out.
6. **findClientByEmail name fallback loads ALL clients into memory.** When the kiosk searches by name instead of email, it does a full table scan. At 2,000+ clients, it downloads every client document to find one match.
7. **No query result limits anywhere.** getClientWithForms loads ALL forms for a client with no limit. A patient who's been coming weekly for 2 years would have 100+ forms loaded in one request. Should have a limit with "load more."
8. **Image uploads are sequential, not parallel.** When a form has multiple images, they're uploaded one at a time - 5 images means waiting for each one to finish before starting the next. They should upload simultaneously. Could cut form submission time in half.
9. **Documents re-read after every write.** updateClient and the SOAP template functions (create, update, duplicate, publish, archive) all write to Firestore, then immediately re-read the same document to return it. Unnecessary - the response can be built from the data already in memory.
10. **Client detail page - ListView anti-pattern (Flutter).** When you open a client's detail page, it loads all their forms into a scrollable list. But instead of only building the forms you can see on screen (say 5-10), it builds ALL of them at once - even the ones you'd have to scroll down to see. If a patient has 100 forms from a year of weekly visits, Flutter creates 100 form widgets the moment you open the page, even though you can only see 5 at a time. That's why it gets laggy - the browser is doing 20x more work than it needs to. The fix: only build forms as they scroll into view. Flutter has this built in (ListView.builder), it's just not being used correctly here.

09. Remediation Plan

Phase I – Must fix before launch

Task	Priority	Effort	Status
Database security rules + audit log immutability	Critical	Medium	Done
Move OpenAI to backend + PHI redaction	Critical	Large	Done
Sign BAAs (Google Cloud + OpenAI)	Critical	-	Done
Audit trail gaps (read logging + strip SSN from logs)	High	Small	Done
Database backups	High	Small	Done
Pagination + denormalization	High	Large	Done
HIPAA policies + Security Risk Analysis	High	Medium	Done
Tests for critical paths	High	Medium	Done
Public form protection (rate limit + CORS)	Medium	Small	Done
Password reset flow	Medium	Small	Done
Error monitoring	Medium	Small	Done
Setup staging environment	Medium	Medium	Done
Deployment & publishing		Small	Done

Phase II – Fix after launch

Task	Priority	Effort	Status
Setup MFA (agreed with the client to do right after launch)	High	Medium	Done
File size limits on uploads	Medium	Small	Done
Admin session - end on browser close + inactivity timeout	Medium	Medium	Done
Password minimum 8+ with complexity	Medium	Small	Done

Task	Priority	Effort	Status
Token revocation on user deactivation	Medium	Small	Done
Remove patient identifiers from logs	Low	Small	Done
Deployment & testing		Small	Done

Roadmap

1. Phase I - fix security/HIPAA issues + core performance issues, make it ready for launch
2. Phase II - fix remaining security + performance / tech debt issues
3. Phase III - refactor before building anything new

Bladeware builds and fixes healthcare software - HIPAA & Security audits, remediation and custom development.

Sample report. Client name and identifying details have been removed.