ChatGPT

FaithSeal SaaS Audit

1. Pro Feature Gating

Backend: All “Pro” features are protected by `checkProClaim`. For example, the **Ethics Analysis**, **Contract Summary**, **Bulk Row Processing**, and **Logic Rules** functions each call `checkProClaim(request, …)` at the start [1] [2] [3] [4]. This throws a permission error if the user lacks `pro_tier` in their auth token [5]. Thus no Free/Faith user can invoke these cloud functions.

However, **ownership validation is missing**: none of these functions verify that the given `docId` actually belongs to the calling user/tenant. For instance, `summarizeContract` will write a summary into any `documents/{docId}` provided, without checking that `documents/{docId}.owner == uid`. This could allow a malicious user to trigger analysis on arbitrary document IDs (though they still supply the content). We recommend adding a lookup on the document (or tenant field) before saving results.

Frontend: The React UI should similarly gate these features. Hooks like `useUserPlanStatus` rely on the `plan_type` and `pro_tier` claims to disable UI. For example, `useUserPlanStatus` sets `planType` based on `claims.plan_type` or `claims.plan` [6]. Feature flags or plan checks should prevent free users from even seeing the Pro widgets. (Documentation hints at this: e.g. a Template Preview modal only shows contract summaries for Pro users [7].) Importantly, the **Ethics Analysis** is shown as “Faith+” in docs [8] but in code *requires Pro*. This inconsistency means the UI might allow Faith-tier users to attempt it, only to get an error from the backend. Ensure the UI matches the backend rule (or update claims logic). Free users should see a clear “Upgrade to Pro” prompt instead of attempting the call.

Observations: No obvious React code was found for `FaithSealAiConsole.tsx` or similar gating components. We suggest reviewing those components (and `BulkUploadForm`, `FormEditor`) to confirm they disable Pro features unless `plan_type === 'pro'`. For example, wrap the Pro-feature UI in something like:

```
if (claims.plan_type !== 'pro') {
  return <UpgradeMessage />;
}
```

and disable buttons.

2. Async AI Task Handling

Currently, the AI features are implemented as synchronous `onCall` functions rather than Cloud Tasks. As such, there is no Firestore “status” field (like `{status: "processing"}`) updated over time. Each function logs progress to Cloud Functions logger (e.g. start/end of API calls) but **does not record status to the document**. For example, `analyzeDocumentEthics` simply sends the request and returns the result when done, and on errors it throws an HttpsError [9]. If an AI call takes time, the client must just await the promise.

- **Ownership Validation:** As above, none of the AI tasks validate that the `documentId` belongs to the user's tenant. This should be added before saving results.

- **App Check:** We see no use of App Check tokens in the AI functions. It would be good practice to verify `request.app` or use `checkApp` to ensure only your frontend can call these.

- **Logging and Status:** The functions do extensive logging (e.g. `logger.info` before/after API calls [10] [11]). However, they do not update any `documents/{docId}/status` field. If the UI needs to show progress or error states, consider adding a Firestore document field like `aiAnalysis.ethics.status = 'processing'/'done'/'error'` with timestamps.

- **Retries and Resilience:** The contract-summary function has retry logic for transient API errors [12], which is good. The ethics analysis has no retry loop – a single failure immediately bubbles up. It may be worth adding a retry or at least catching timeouts (the Gemini API can be flaky). In all cases, failures are logged and thrown so the client sees an error (e.g. [`throw new HttpsError('internal', ...)` at [9]]).

- **Saving Results:** Notice `summarizeContract` checks if the target document exists and even looks up by `externalId` [11] before saving. This prevents silently writing to a non-existent doc. In contrast, `analyzeDocumentEthics` just does a `.set()` on the subcollection [13] without checking. It may auto-create the doc placeholder. Align both functions to check existence like summary does.

- **Frontend Reflection:** Since these are instant RPCs, the React UI will simply show a spinner until the function returns. No explicit "in-progress" state is tracked. That is OK, but test cases should include slow network/Gemini delays. Also test what happens if the Cloud Function crashes or returns an error: the UI should catch and show an error message rather than hanging.

3. CI/CD Hardening

The GitHub Actions workflows are generally well-configured: they use pinned action versions (e.g. `actions/checkout@v4`, `ruby/setup-ruby@v1`, `actions/setup-node@v4`) [14] [15]. Secrets (like `FIREBASE_TOKEN`, Stripe keys) are read from GitHub Secrets or GCP Secret Manager, and not hard-coded. Dependency scanning is in place: bundler-audit and `npm audit` are used in **improved-dependency-scan.yml** (we saw `bundle-audit` and `npm-audit` invoked) [16] [17], and codeql SARIF uploads are configured. The `functions.yml` deploy job uses `$FIREBASE_TOKEN` [15], which is correct.

Recommendations: Consider adding image scanners or signing if Docker is used. (The repo has a Dockerfile, but it's not built in CI here). A Trivy scan or Cosign signing step could be added when building the `faithseal/faithseal` Docker image. Also ensure any new Actions are pinned by SHA or version.

4. Rails → Firebase Sync

- **Claim Syncing:** Subscription state is synced via Stripe webhooks and scheduled functions. For example, the Stripe webhook handler logs events to Firestore and calls `updateUserClaimsSafely` (in the imported `claimsManager`). It then updates the user's `pro_tier` and `plan_type` claims. After Stripe events, you should verify in the Firestore `users/{uid}` doc that `proTier` matches the plan in Stripe. The `syncProClaims.ts` function even scans for mismatches [18]. Ensure that "manuallySyncUserProClaims" (if it exists) enforces the same logic. If a Rails admin triggers a manual sync, it should call the same Firebase function (or update the Firestore `users` doc appropriately).

- **Stripe Plan Syncing:** The Cloud Function `stripeWebhookHandler` processes subscription events and writes logs to `subscription_events` [19]. This is good for auditing. Verify that the Stripe Price IDs in env match your plans (see `STRIPE_PRICE_PRO_MONTHLY` env in `syncProClaims.ts`). The code maps Stripe price IDs to plan names [18]. You should test that when a user changes plan in Rails (if Rails calls Stripe) or in Stripe Portal, the webhook updates Firestore.

- **Bulk Job Status:** Currently, `processBulkRow` (for CSV processing) runs per-row via Cloud Functions. There is no explicit "bulk job" document tracking progress in the code we saw. On Rails side, ensure that the admin UI can poll or receive updates about bulk import progress (maybe via Firestore triggers or by checking a `bulkJobs` collection). If not implemented, consider adding a Firestore `bulkJobs/{jobId}` doc that is updated as rows succeed/fail, so the Rails admin can display progress.

5. Testing Gaps & Edge Cases

- **Auth Tests:** Write tests for calling each Pro function with (a) no auth (expect UNAUTHENTICATED) and (b) auth but free-tier plan (expect PERMISSION_DENIED). For example, call `summarizeContract` with a free user token and assert an error message "Pro plan required…" [5]. Also test expired or cancelled subscriptions by simulating a user whose `pro_tier` claim was removed.

- **Tenant Isolation:** Test that a user cannot read or write another tenant's data. For instance, attempt to save an AI result to a `docId` owned by someone else. Since rules are locked down, the function will still run (due to admin privileges) but the Rails/Next UI should prevent showing foreign docs at all.

- **Token Propagation:** There can be a delay between a user upgrading in Stripe and the new claim appearing in `request.auth.token`. Test scenarios where the user upgrades and immediately attempts a Pro feature: ensure the claim sync job runs quickly, or consider short-circuiting by calling `setProClaim` manually.

- **Race Conditions:** In multi-step processes (if any Cloud Task chain exists), simulate failures at each step. Ensure that if step 1 succeeds but step 2 fails, the system can retry step 2 without duplicating step 1 effects.

- **Token TTL:** Firebase auth tokens cache custom claims for up to 1 hour by default. Test the edge case where a user's plan downgrades and their token still has `pro_tier = true` for a bit. The `enforceSubscriptionClaims` function should handle grace periods, but in the UI, the user could temporarily access Pro features. Be aware of that lag.

- **Bulk Processing:** If CSV uploads spawn multiple `processBulkRow` calls, test with very large CSVs. Ensure that hitting concurrency limits does not crash Cloud Functions. The bulk functions have concurrency=10 and 120s timeout [20]; test that large row data fits within memory/cpu.

Confidence Ratings

- **Backend Auth (Functions)**: High. All Pro functions **do** enforce `pro_tier` [1] [2]. (*Gap:* no ownership check).
- **Frontend Gating**: Moderate. React hooks are present for plan status [6], but we did not inspect all components. Ensure every Pro-widget checks the user's plan before enabling.
- **Firestore Rules**: Low/Locked. The `firestore.rules` fully denies client access to everything [21]. This is "safe" but means **all** data access must go through functions or admin SDK. If that is intended (only trusted code accesses Firestore), it's fine. No rules allow end-user read/write.
- **CI/CD Security**: High. Workflows use pinned actions and include Linters (ESLint, RuboCop) and scanners (`bundle-audit`, `npm audit`). No obvious leaked secrets in code. (As a note, rotate any placeholders in `SECURITY_NEXT_STEPS.md`.)
- **Async Task Logic**: Medium-Low. Since the AI features aren't truly async (no Cloud Tasks), most logic is single-shot. The existing logging and retry help, but we'd score this lower due to missing explicit state tracking and some missing validation (no doc ownership, no App Check).

Sources: Internal code and docs were audited. Key references include the Pro-claim checks in our Cloud Functions [1] [2] [3] [4], the plan-determination hook [6], and subscription mapping in Stripe sync code [18]. The subscription docs also list which features are gated at Faith vs Pro tier [8]. These guided the above observations and recommendations.

---

[1] [9] [10] [13] github.com
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/functions/src/ai/analyzeDocumentEthics.js

[2] [11] [12] summarizeContract.js
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/functions/src/ai/summarizeContract.js

[3] [20] github.com
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/functions/src/bulk/processBulkRow.js

[4] github.com
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/functions/src/logic/evaluateLogicRules.js

5 github.com
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/functions/src/utils/claimValidation.js

6 useUserPlanStatus.js
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/app/javascript/hooks/useUserPlanStatus.js

7 github.com
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/AI_INTEGRATION_README.md

8 SUBSCRIPTION_FEATURES.md
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/SUBSCRIPTION_FEATURES.md

14 ci.yml
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/github/workflows/ci.yml

15 functions.yml
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/github/workflows/functions.yml

16 17 improved-dependency-scan.yml
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/improved-dependency-scan.yml

18 syncProClaims.ts
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/functions/src/auth/syncProClaims.ts

19 stripeWebhookHandler.ts
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/functions/src/payments/stripeWebhookHandler.ts

21 firestore.rules
https://github.com/Tjdolan-ai/faithseal-clean1/blob/9d390368e5aa6e5bbdd5087afa4c5b944ff58a8b/firestore.rules