Mantifang Constellation - Privacy & Knowledge Ownership Audit

Mantifang Constellation

Date: 8 October 2026 Conducted by: Claude (independent code audit, no implementation changes made) Method: Static analysis of the actual implementation - the backend code, database migrations, the frontend code, configuration files, and the existing test suite - not based on assumptions from documentation. Where the project's own documentation disagreed with what the code actually does, the code was treated as ground truth, and the discrepancy itself was recorded as a finding. No file in the codebase was changed during this audit. No destructive actions were taken, no real personal data was used, and no production system was touched.

How to read this report

Every finding below is classified as one of:

- PROVEN - demonstrated directly in the code, in some cases backed further by the existing automated test suite.
- RISK - a concrete, technically grounded problem that has been identified.
- UNKNOWN - there isn't enough evidence to settle the question one way or the other; it would require investigation beyond what this codebase can show.
- MISSING - the functionality in question simply does not exist in the code.

Nowhere in this report is the absence of evidence treated as evidence of safety.

1. Executive summary

In practice, Mantifang Constellation is two systems running in one process: Hugo's own, single-tenant writer's workbench, and a separate multi-tenant Frontrunner/Member platform with its own session-based authentication, a public gateway at app.shikibu.app, and its own private workbench and library for every member. The project's existing architecture and security documentation describes neither of these systems accurately - that documentation is demonstrably out of date and should not be relied on for any public claims.

The core promise, "the knowledge is yours," is for the largest and most fundamental part technically well-founded: a Frontrunner's private workbench and library content is structurally excluded - through a separate database table, never a query that could accidentally reach another person's row - from Ask Shikibu, from the shared knowledge base, and from every form of external AI. That is strong, verifiable evidence, not a marketing claim.

At the same time, there are concrete gaps that mean the full promise is not yet kept today:

- There is no complete, self-service data export. A Frontrunner cannot, in a single action, download all of their own workbench and library data together with their images. (MISSING)

- Knowledge once contributed to the Constellation cannot be withdrawn. There is no "undo" for an already-approved contribution, and a submission that is still pending review survives the deletion of the member's own library source it was drawn from. (MISSING / RISK)

- The classification of what counts as "public" in the shared knowledge base is fail-open: a document is treated as public unless it carries one specific internal marker, rather than the reverse. This is a structural risk, not (so far) a demonstrated active leak of Frontrunner content specifically. (RISK)

- No form of content detection exists anywhere in the codebase - no virus scanning, no abuse-material detection, no moderation of any kind. (MISSING - see the separate chapter on this, section 7.)

- The actual internet exposure of app.shikibu.app runs through infrastructure outside this codebase, and could not be audited from the code itself. (UNKNOWN)

Final verdict: CONDITIONAL GO - see section 10 for the full reasoning and the conditions attached to it.

2. Architecture and data-flow overview

(Compiled from direct investigation together with a focused review of all nine requested data flows; every point below was personally checked against at least one piece of evidence in the code, never taken on faith.)

2.1 The personal Workbench (Frontrunner)

A Frontrunner's personal workbench items and any images attached to them are stored in their own dedicated tables. Isolation is proven: every read, write, and delete query is explicitly scoped to the acting user's own id and the specific item id - documented in the code itself as a deliberate design choice, not an accident. Image uploads are possible only for adult accounts; a child can write ordinary text notes, but cannot upload a single image, in any note type.

2.2 My Library (Frontrunner)

The library uses the same user-and-item scoping pattern as the workbench. An item only ever leaves a member's private scope through one explicit, member-initiated action - contributing it to the Constellation - never automatically, and never as a side effect of any other action. Deleting a library item never touches the separate Constellation record that an earlier contribution from that same item may already have created - see section 3.4 for what that means in practice.

2.3 The database and file storage

Hugo's own workbench and the entire Frontrunner gateway share one single, unencrypted database file, running in the same process on the same server. Isolation between members is purely logical - enforced by WHERE-clauses in every query - never physical; there is no separate database file or schema per member. Large imported source files (PDFs, EPUBs, images, including approved Frontrunner contributions) are stored on disk under a content-addressed path, keyed by a cryptographic checksum of their own content, capped at 500 MB per file. Backups are full, unencrypted copies of the entire database - which means, by definition, that every backup contains

every Frontrunner's private data in one single file, with the same file permissions as the main database.

2.4 Shikibu and the local AI model

Core inference runs locally, on-device: the local model loads and runs entirely in-process, with no network call anywhere in that path - confirmed by searching the model-loading code itself for any trace of an outbound request, an API key, or an endpoint string, and finding none. "Ask Shikibu" draws its candidate material exclusively from documents already marked public in the shared knowledge base, plus contributions that already carry an explicit contributor stamp from an approved submission. There is an explicit, structural guarantee written directly into that module's own code: no public-facing Ask Shikibu code path may ever construct a connection to an external AI service - that capability simply has no entry point there, and is declared in the code as something that will intentionally never be added. No table holding private workbench content is ever read on this path at all - not a filter that could be forgotten, but a table that is never touched. This is proven, and proven strongly.

2.5 External AI services and APIs

The module that talks to external AI services (Perplexity, Gemini) is exclusively Hugo's own, personal research tool, authenticated with his own private credential. It has one single choke point through which every call passes, which logs the provider, the type of task, and whether it succeeded - but never the prompt, the response, or the credential itself, confirmed by reading that logging code directly. No file anywhere in the entire Frontrunner-facing part of the codebase imports or calls that external-AI module - confirmed with a targeted search across that whole family of files, which returned zero matches. The Frontrunner-facing equivalent, an "external internet research on/off" toggle, does nothing more on submission than open a new browser tab to a search URL on the member's own device - a client-side browser navigation, with no backend call and no data sent to any external AI provider at all.

Conclusion - proven: a Frontrunner's private knowledge does not reach any external AI provider through any code path that could be found. What Hugo sends to Perplexity or Gemini through his own research tool (for example, to find further reading for his own articles) is always his own content, never a Frontrunner's.

What remains genuinely unknown, and outside what this codebase can answer, is whether Perplexity or Google themselves use the prompts sent to their own APIs for training - that is a question about those providers' own terms of service, not something this code can guarantee or prevent, only avoid sending private data to in the first place, which is already the case here for Frontrunner data.

2.6 Knowledge base and search

See section 2.4 for the hard separation. What actually counts as "public" is examined in section 3.6 below - and that is where a structural risk was found.

2.7 Publishing to the websites

Publishing follows a staged workflow - candidate, then approved, then published - with a row-level guard that requires a candidate to already be in "approved" status before a publish action can proceed at all; a publish can never happen directly from "candidate." Member data enters this pipeline only through the explicit "add to the Constellation" opt-in described in section 2.2 - never automatically.

"Absolute privacy" is enforced through one single, central guard that is called on every single outgoing route - publishing, contributing, becoming a weekly candidate. One important nuance, backed by the code itself: this only blocks future, owner-triggered outgoing actions - it is not an access control against the administrator or the filesystem, and it explicitly does not block a properly authorized Safety Access case (see section 7.3): the code's own documentation states plainly that reaching an "absolute private" item through a fully authorized safety case is not a bypass of that protection, by design.

2.8 Synchronization, caching, logging, and backups

No cloud synchronization of any kind was found - a targeted search for iCloud, Dropbox, S3, rclone, or any remote rsync target turned up only the platform's own social-media "Connect" integrations, no cloud storage of workbench data. Logging, on failure, records only the HTTP method and path of a request - never the request body; confirmed by checking that no logging call anywhere in the main API file passes a request body through. One low-severity, not demonstrated as exploitable risk: unexpected exceptions are returned as raw error text directly in the HTTP response to the requester themselves. In theory, a sufficiently unusual error message could leak a fragment of server state if an exception text happens to interpolate user content - this was not checked exhaustively across every single place an error could be raised. Backups: see section 2.3.

2.9 Mobile clients and authentication

The mobile app configuration enables the settings needed to keep the session cookie alive inside the iOS in-app browser - a known, previously fixed bug. The session token itself is a long, cryptographically random string, stored in an HttpOnly cookie that JavaScript can never read; only a one-way hash of the token is kept on the server side, so a database leak alone would not yield a usable session.

One risk, with an important nuance: the default session cookie deliberately does not carry the "Secure" flag - the code's own comment reasons explicitly that this server runs over plain HTTP, including on the local network. Only the separate cross-origin, packaged-app path receives "SameSite=None; Secure." The problem is that the runtime configuration also lists a real, publicly routable HTTPS domain - app.shikibu.app - for ordinary browser-tab logins, not only the packaged app. For those visitors, the session cookie travels without the Secure flag, even though the endpoint they are actually reaching is HTTPS.

The single most important open question in this entire report: the Python gateway server itself demonstrably binds only to the local loopback address. app.shikibu.app must therefore be exposed to the internet through a reverse proxy or tunnel that lives outside this codebase entirely. The project's own network setup notes explicitly warn against configuring reverse proxies, cloud tunnels, or public DNS - language that appears to be about Hugo's own daily workbench, not this deliberately public-facing Frontrunner gateway, but that outer infrastructure layer - TLS termination, any firewall or rate-limiting, the patch status of the proxy itself - cannot be audited from this codebase at all. This needs to be verified separately, outside this repository, before any claim of being "safe for 1,000 users over the internet" can be supported.

Not tested in this section: no live server or network was actually set up; everything above is static code analysis plus reading the existing test suite. The real reverse proxy or tunnel configuration for app.shikibu.app exists outside this repository and was not examined. The runtime behavior of the public-document classification under a corrupted or missing marker was only read statically, never

reproduced dynamically against a test database.

3. Privacy and ownership assessment

3.1 Access limited to one's own private data - proven, with one representative-sample limitation

One central authorization check gates every route that touches a specific member's own data. Object-level scoping - matching both the acting user's id and the specific item id - was confirmed for the most important routes. Three routes deliberately bypass that central gate and verify access themselves internally; all three were checked and found correct. Stated limitation: this was a representative sample of the highest-risk patterns, not a line-by-line review of every route in the roughly 440-kilobyte main API file. For the routes not directly checked, this remains unknown, not proven safe.

3.2 Unauthorized access by other members, administrators, or AI processes

Other members: proven safe (see 3.1). The administrator, Hugo, through the application itself: proven safe - the admin-facing "Members" view deliberately shows only counts of workbench and library items, never their titles or contents, with an explicit comment in the code stating there is no reason to surface a Frontrunner's private content to Hugo just because he technically could query the database. The administrator through the filesystem itself is a structural limitation, not a bug: Hugo, as the operator of the Mac the platform runs on, has physical and operating-system-level access to the one unencrypted database file described in section 2.3. The application layer cannot prevent this - it is inherent to a single-operator, self-hosted architecture without database encryption, not a flaw in the code. This must be stated as an explicit limitation in any public claim, never left unsaid (see section 8). AI processes: proven safe (see sections 2.4 and 2.5).

3.3 Unintended appearance in shared knowledge bases or search results

See section 3.6 below - a structural risk (fail-open classification), though not a proven active leak for Frontrunner content specifically, since the one confirmed caller that ingests Frontrunner contributions is correctly tagged.

3.4 Consent, withdrawal, and "contributing to the Constellation"

Contributing is one-way and irreversible - missing: there is no retraction function. Once Hugo approves a contribution, there is nowhere in the codebase a function that removes an already-shared document from the Constellation again at the original contributor's request. A concrete, reproducible risk: a pending submission stores a full copy of its own content separately, in its own table. Deleting the originating library item never touches that separate copy. A Frontrunner who deletes their own library item after already submitting it for review withdraws nothing by doing so - Hugo can still approve the submission from that separate, never-cleaned-up copy. There is also no self-service "withdraw my submission" action while it is still pending. Additionally missing: there is no retention or cleanup policy for the stored content of a rejected submission - rejecting it only changes its status; the raw content stays in the database permanently.

3.5 Export and deletion

Deletion is largely proven, with one explicit and sensible limitation. Permanently deleting an account

removes the account and all directly linked personal data - workbench items and their images, library items, sessions, password-reset tokens - plus two related tables that are cleaned up explicitly in code rather than through database cascades. The function deliberately refuses to proceed wherever any form of shared history exists - child accounts, calls, projects, an open safety case, a parental statement - with a readable reason given each time. That is a deliberate, defensible design choice, not a bug. What this function explicitly does not remove, confirmed in its own documentation, is a document that has already been contributed to the shared Constellation - consistent with, and a direct consequence of, the finding in 3.4.

Export is missing: there is no complete, self-service export. The only export function found exports a single library item as an HTML file. No function was found that collects all of one Frontrunner's workbench items, library items, and attached images together in one action. A claim such as "you can take all your knowledge with you at any time" is not demonstrable today.

3.6 Fail-open classification of "public" knowledge-base content - risk (structural)

This is the single most important independently found finding of this report, not something flagged by the background research that fed into it.

The function that decides whether an ingested document counts as public is called on every ingest to set that document's public/private flag. What it actually does is return "public" for everything, unless the document's title, slug, raw content, or metadata contains one of a short, specific list of internal marker phrases - things like an internal "noindex" marker, or an explicit internal-workflow or editorial-control tag. This is a fail-open classification: a document is public unless it is marked otherwise. The safer, privacy-by-design direction would be the reverse: private unless explicitly marked public.

Why this is not a demonstrated active leak for Frontrunner content today: the one verified caller that ingests Frontrunner contributions explicitly sets exactly one of those excluded internal markers. For that specific path, it works correctly, today.

Why it is still a real risk: any future, or not-yet-found, ingest call that forgets to set that marker - a typo, a new contribution path, an edge case where metadata is empty - becomes silently public instead of silently private. A fail-safe architecture would want this the other way around: a forgotten marker should lead to something being over-protected, never to a leak. This was not reproduced dynamically - doing so would require a modified test database, which fell outside the "no real personal data, no destructive actions" boundary of this audit - so it remains a risk, not a proven active leak.

4. Findings, summarized

1. Fail-open classification of public documents. (RISK)
2. The session cookie lacks the Secure flag on the public HTTPS endpoint. (RISK)
3. The internet exposure of app.shikibu.app runs through infrastructure outside this codebase that has not been audited. (UNKNOWN)
4. There is no way to retract an already-approved Constellation contribution. (MISSING)
5. Deleting a library item does not stop a still-pending submission drawn from it. (RISK)
6. There is no cleanup of a rejected submission's stored content. (MISSING)
7. There is no complete, self-service data export. (MISSING)

8. Raw exception text is returned in HTTP error responses. (RISK, low severity)

9. There has been no exhaustive, route-by-route authorization check of every route in the main API file. (UNKNOWN)

10. No form of content detection or moderation exists anywhere. (MISSING)

11. There is no self-service way for a Frontrunner to report another member or specific content. (MISSING)

12. The project's core architecture and security documentation does not describe the Frontrunner system, or is aspirational and out of date. (RISK, governance)

Positive findings, backed by evidence, not merely an absence of problems:

- Passwords are hashed with PBKDF2-HMAC-SHA256 with a random salt; session tokens are 32 random bytes, with only a hash stored server-side.

- Admin preview sessions are structurally read-only - every write action is blocked whenever a session is flagged as a preview.

- Shikibu's retrieval is structurally - by table, not by filter - cut off from private workbench content (section 2.4).

- "Safety Access" is an already-existing, well-considered, scoped, time-limited, two-person-controlled inspection mechanism - see section 7.3.

- Image uploads are blocked for every child account, consistently enforced.

5. Comparison with commercial AI assistants

(An architectural comparison based only on what was demonstrated in this codebase - no unproven claims about specific competitors.)

On control over core processing: Mantifang Constellation defaults to a local model, with no network call anywhere in the Shikibu path, whereas a typical commercial cloud AI assistant processes everything on the provider's own servers by design.

On dependence on external AI providers: here, it is optional, can be switched off entirely, used only for Hugo's own editorial work, and never for Frontrunner data. A typical commercial assistant is often the only processing route available at all - without the provider, there is no functionality.

On reuse of data for training: not demonstrated for this platform itself - no training pipeline was found in the code. For the external providers themselves, this remains unknown and outside the scope of this audit. For a typical commercial product, this varies by provider and plan, and is usually a matter of the provider's own terms of service, not something a user's own code can verify.

On ownership of the data: here, one local database file, owned by the operator, with no cloud storage found. A typical commercial product usually stores data centrally, at the provider.

On exportability: here, partial - per item, not complete. Large commercial providers often offer a complete export, sometimes because they are legally required to.

On transparency of the source code: here, fully visible - this audit itself was only possible because the code can be read. A typical commercial product is usually closed-source, with behavior that cannot be independently verified.

On autonomy of publication: here, an explicit human-approval layer is demonstrated everywhere -

staged workflows, an "absolute privacy" guard. This varies widely across commercial products.

Conclusion: on local processing, no external AI dependency for Frontrunner data, and transparency, Mantifang Constellation is demonstrably better positioned than an architecture that inherently relies on an external cloud AI service. On complete exportability and on formal, auditable governance - a published privacy policy, terms of service with an explicit prohibition on illegal content - the platform is not yet at the level of a mature commercial product.

6. Prioritized remediation plan

Before any scaling up (P0): verify and document the actual internet exposure of app.shikibu.app - the reverse proxy or tunnel in use, its TLS configuration, and who manages it.

P0: Change the public-document classification to fail-closed - private or internal unless explicitly marked public - together with a regression test that expects a document without a marker to be treated as private.

P1: Set the Secure flag on the session cookie whenever a request arrives over HTTPS, via a forwarded header from the proxy.

P1: Build a complete, self-service export function covering all workbench items, library items, images, and profile data.

P1: Tie deleting a library item to automatically withdrawing or locking any still-pending submission drawn from it, and add an explicit "withdraw my submission" action.

P1: Add a retention policy for rejected submissions, clearing their stored content after a fixed period.

P2: Investigate and document, with a formal decision record, whether and how an already-contributed document could be withdrawn from the Constellation on request.

P2: Return a generic "internal error" message to the client, without the raw exception text.

P2: Add a self-service way for a Frontrunner to report a concern, feeding into a Safety Access case.

P2: Update the project's core architecture and security documentation so it describes the system as it actually is today.

P3: An exhaustive, route-by-route authorization check of every route in the main API file - potentially automated, using a script that checks every per-user route against a second test account.

P3: CSAM-specific measures - see the separate remediation plan in section 7.6.

7. Separate investigation: could private workbenches be misused to store or distribute illegal material, including CSAM?

(Investigated independently and thoroughly; no destructive actions were taken, no real files were uploaded - this is based entirely on analysis of the existing upload, storage, and review code paths.)

7.1 The two relevant storage paths, mapped precisely

A. Private note images ("+ Photo", workbench notes). Only adult accounts can use this path at all - child accounts cannot structurally upload any image, in any note type. Each image is capped at two million bytes once decoded. There is a content-type allowlist and base64 validation, but no actual image-structure validation - the bytes are never decoded as an image to confirm they really are a valid

image, so an arbitrary binary file carrying a valid image content-type label would pass through. That is a separate, smaller technical point - not itself a CSAM-specific risk, but relevant to any future automatic detection, which would itself need to be robust against invalid "images." Storage is a text column in the database, never a separate file on disk, strictly scoped to the specific member and item - confirmed in section 3.1. There is no form of review, scanning, or approval on this path at all - this is precisely the "private stays private" design, and so it is also precisely the blind spot for this specific risk.

B. "Submit sources" - file or URL submissions to the shared Constellation. This path accepts a wide range of file types, including raw image files, up to 500 megabytes. The complete, raw bytes are stored as a pending submission directly in the database, not on disk. Only on approval - reachable exclusively through a loopback-only administrative route - are these bytes finally written to disk and made part of the shared knowledge base, through the platform's own canonical ingestion process. On rejection, the raw bytes remain permanently in the database (see finding 6 in section 4) - not in itself a lasting evidence problem (see 7.5), but a missing explicit policy.

Crucially, and positively: this path already has a mandatory human review step - Hugo, admin-only - before anything is ever shared or permanently written to disk as a file. This was not built for CSAM moderation, but it already functions as a de-facto, pre-existing checkpoint for anything a Frontrunner wants to bring outside their own private space.

7.2 What does not exist

A targeted search across the entire backend for any trace of moderation, CSAM handling, NSFW detection, perceptual hashing, PhotoDNA, NCMEC, abuse reporting, content scanning, or virus scanning returned zero matches, aside from one unrelated use of the word "moderate" as a confidence level elsewhere in the code. Entirely missing:

- Automatic detection - no hash-matching against known CSAM hash databases, no machine-learning classification, nothing of the kind.
- No virus scanning of uploads.
- No quarantine mechanism for suspicious files.
- No self-service way for a Frontrunner to report another member or specific content - a targeted search for reporting functionality turned up only the administrator side, never a member-facing button.

7.3 What already exists - and how strong it is

"Safety Access," built independently of this audit, is already a heavily gated, fully audited inspection path into private workbench content, and comes very close to exactly the kind of proportionate mechanism this investigation was asked to look for.

It has three governance levels. For adult members: Hugo may open a case only after strong re-authentication through a dedicated security key, every inspection is fully logged, and Mickey - an existing, ordinary Frontrunner account repurposed as "Safety Reviewer" - must review it afterward. For child accounts: both Hugo's security-key re-authentication and Mickey's separate, case-specific approval are required before a case can even become active - genuine two-person control. For law enforcement: this module is never involved at all - no account, no API, no automatic handover, a deliberate design choice.

The scope is minimal by default - only the originally reported resource; viewing anything beyond that requires an explicit, separately audited action with a stated reason. It is time-limited to thirty minutes, with no automatic extension. It keeps a complete, append-only audit trail - no route can ever modify or delete a logged case event. And it is technically capable of actually showing a specific reported workbench item, including its images, to Hugo - within a tightly scoped, fully audited case.

The important limitation of this existing mechanism, specifically for CSAM, is that it is entirely reactive. It provides no proactive detection at all - if nobody opens a case, and as established there is no self-service reporting button for an ordinary member, any image inside a private note stays structurally unseen, forever, exactly as the privacy design intends. That is the core of the blind spot.

7.4 Legal context (not legal advice - for orientation only)

In the Netherlands, article 240b of the Criminal Code prohibits producing, distributing, offering, and possessing child-abuse material; a hosting party has a duty of care, once made aware, to remove such material and report it, typically through the Dutch INHOPE-affiliated hotline. In the EU, the temporary CSAM regulation permits - and in some proposals requires - providers to detect and report such material on a voluntary, proportionate basis, typically through hash-matching rather than human review of all content. In the United States, where applicable, federal law requires an electronic communication service provider to report to NCMEC's CyberTipline upon actual knowledge of CSAM, with specific requirements for preserving evidence. Which of these actually applies depends on where the platform is legally established, where its servers and infrastructure are located (the still-unresolved question from section 2.9), and where its users are - a question for a lawyer, not something a code audit can settle. It is included here so that it is explicitly on the agenda, not as definitive legal advice.

7.5 Proposal: proportionate prevention and handling, without blanket access

The starting point - no general human access to private files - is already largely the current architecture. The proposal below deliberately builds on the existing Safety Access mechanism rather than introducing a parallel system.

First, automatic, hash-based detection in place of human review: on every upload, compute a perceptual hash - for example through a recognized partner service, or after formal registration, an official hash set - and compare it against a database of known CSAM hashes. This requires no human look at the image itself for the overwhelming majority of, non-matching, cases - only a match triggers a next step. This is the industry standard precisely because it allows detection without "looking" at content. It requires a formal registration process with the hash provider - a process and legal step, not purely an implementation one.

Second, a match should automatically open a Safety Access case, rather than relying on Hugo's manual trigger - for example a new case category alongside the existing "suspected abuse" one, with the same existing two-person, security-key-gated governance applying unchanged. This keeps the existing, already-audited minimal-scope and time-limited guarantees fully intact - no new, blanket form of access.

Third, quarantine, never silent deletion and never silent retention: on a confirmed match, make the item immediately inaccessible to the uploader, without telling them why - consistent with standard practice to avoid destruction of evidence - preserve the evidence for the legally applicable period, and do not independently delete it before a report has been made, since in multiple jurisdictions premature deletion without reporting is itself a problem.

Fourth, a documented reporting and escalation protocol, separate from the code: who calls or emails which hotline, who within the organization has the authority to act - at a one-person operation, Hugo himself - and how this gets recorded in the existing, unchangeable case-event log, which is already suited to this purpose.

Fifth, a self-service way for a Frontrunner to report a concern, which, after a brief triage, feeds into the same Safety Access case structure. This closes the "nobody ever opens a case" blind spot for content Hugo himself never sees but another member happens to encounter, for example shared within a joint project.

Sixth, a retention and cleanup policy for non-matching, rejected submissions: after a fixed, short period, delete the raw stored content of a rejected submission that never led to a match or a report - content that never triggered a match or a report should not sit in the database indefinitely.

Seventh, explicit inclusion in the platform's terms page: a clear, user-readable prohibition on illegal content, including CSAM, with a reference to the reporting procedure - today, this specific prohibition is missing from the existing sections.

7.6 Prioritized remediation plan, CSAM-specific

P0: Write and document the reporting and escalation protocol - costs no code, but is needed immediately. P0: Add the explicit prohibition and reporting method to the terms page. P1: Register with a hash-matching partner and implement automatic detection. P1: Tie a match to an automatic Safety Access case, plus quarantine. P2: Add a self-service reporting button for Frontrunners. P2: Add a retention policy for rejected, non-escalated submissions.

Not tested in this section: no file was uploaded at all, not even a harmless test file - the findings above rest entirely on reading the upload and storage code, not on an actual end-to-end upload attempt. A safe, non-destructive click-flow test with a harmless test file could confirm the paths described here dynamically, but was not carried out in this round, since code analysis alone already established this with sufficient confidence. No contact was made with a hash-matching partner, a hotline, or a lawyer - that falls outside what a code audit can or should do.

8. Claims Mantifang can responsibly make publicly, today

1. "Shikibu runs locally by default; your conversations with Shikibu never go to an external AI provider." - Proven, section 2.4.
2. "External AI services are used only by Hugo himself, for his own editorial research, never with your private data, and you can rely on this being off unless Hugo explicitly switches it on for his own work." - Proven, section 2.5.
3. "Your private workbench and library content is technically separated from what other members or the public website can see - not just a setting, but a separate piece of code that never reads this data at all." - Proven, sections 2.1 and 2.4.
4. "None of your private content appears in the shared Constellation knowledge base or on any website without you explicitly choosing to put it there." - Proven, sections 2.2 and 2.7.
5. "You can have your account and your private data deleted." - Proven, with the added, honest nuance that this is refused while shared history is still attached (section 3.5) - a nuance that must be included in the claim, never left out.

9. Claims that cannot yet be made (and should not be made today)

1. "You can always fully export all your knowledge." - Missing, section 3.5.

2. "You can always withdraw a contribution to the Constellation again." - Missing, section 3.4.

3. "Nobody, not even Hugo, can ever access your private data." - This cannot stand as an absolute claim. At the technical, application layer, it is proven to be shielded. At the physical, operator level, Hugo has filesystem access to the single, unencrypted database, and there exists a deliberately limited and fully audited inspection mechanism for concrete safety cases (section 7.3). The honest claim is: "your private data is never shown to others by the application, and is only ever viewed by Hugo in a heavily regulated, fully audited exceptional case."

4. "This platform is fully secure against unauthorized access over the internet." - Unknown, section 2.9 - the infrastructure that actually exposes app.shikibu.app cannot be verified from this codebase.

5. "There is no risk whatsoever of misuse for illegal content." - See the full CSAM chapter, section 7 - no detection mechanism exists today.

10. Final verdict

Question: Is Mantifang Constellation technically ready to give 1,000 Frontrunners access, without their personal knowledge being usable or distributable in an uncontrolled way?

Verdict: Conditional go.

Reasoning: the fundamental promise - that private knowledge is structurally shielded, at the code level, from other users, from the public knowledge base and search, and from every external AI - is demonstrably true, not merely stated. That is the heaviest technical bar, and it is met. The existing Safety Access governance is unusually mature for a project at this scale.

An unconditional go is not responsible, however, as long as:

- the actual internet exposure of app.shikibu.app cannot be verified from the codebase (section 2.9) - at 1,000 users, this stops being a theoretical question;

- the public-document classification remains fail-open (section 3.6) - a small mistake in a future change could already silently publish content that should have stayed private;

- no form of CSAM detection exists at all (section 7) - at 1,000 users, the chance of misuse increases, and the current protection is entirely reactive, dependent on someone happening to report it, and there is not even a reporting button;

- there is no complete data export and no way to withdraw a contribution (sections 3.4 and 3.5) - relevant both to trust and, where applicable, to users' legal rights.

None of these points reflect carelessly built code - on the contrary, the architecture found here is, on almost every other point, deliberate and well thought through. These are concrete, bounded, solvable gaps between the current state and the scale that 1,000 users requires. The priority-0 and priority-1 actions in sections 6 and 7.6 are the conditions for an unconditional go.

- End of report. No further action was taken, per the scope of this assignment: stop after delivery.