ANIMÖLOGIC

TECHNICAL BRIEFING

Technical Briefing

Behavioral Claims Mapped to Deployed Code

Issued by	AIVG Holdings LLC
Contact	info@aivgholdings.com
Platform	animologic.com
Date	July 2026 — Revision 3

Companion: animölogic — The Journey

> The codebase is the authority. This document is a guide to it.

Preface

Each section below corresponds to a claim in the animölogic press release. For every claim, this document names the specific file, function, or route that implements it, and describes what a reviewer would find there. The codebase is the primary source. This document is a map to it.

No claim in this briefing is stated without a corresponding code location. If a claim is not here, it is not in the press release either.

This is revision 3. Claims 01 through 18 are unchanged from revision 2. Claims 19 and 20 are new, covering the replacement of all browser prompt dialogs with inline forms and the three documented re-entry paths shipped in the July 2026 update.

Claim 01 — Consent Gate

Two consent checkboxes must both be checked before the button becomes active. Neither box is pre-checked. The button is disabled in HTML before any script runs.

`client/index.html` — threshold section • `server/animologic-routes.ts` — POST /api/consent/:channel

The two checkboxes carry `id="env-agree"` and `id="obs-agree"`. Neither has a `checked` attribute. The "Accept and approach the seam" button carries `id="env-enter"` and the attribute `disabled`. A JavaScript listener on both checkboxes calls `updateEnter()`, which sets `envEnter.disabled = !(envAgree.checked && obsAgree.checked)`. The button cannot become active unless both boxes are checked.

When accepted, the server route writes a `ConsentedTo` relationship to Neo4j with three properties: channel, version, and timestamp. No name, email, or identifying information is written at this step.

Claim 02 — The Seam

The seam presents three questions. Any answer of sufficient length is accepted. The questions are not scored.

`client/index.html` — seam section

The seam renders three questions in sequence. Each question has a textarea and a Continue button. The Continue button is disabled until the textarea contains at least one character. There is no scoring function. There is no minimum length beyond a single character. The

answers are passed to the reflection engine after all three are answered. Nothing about the answers determines whether the person proceeds. The seam is a passage, not a filter.

Claim 03 — Reflection Prompt

The reflection does not soften or reframe what was said. The LLM prompt instructs it explicitly: "Do not make it sound better than it is. Do not add encouragement."

`server/animologic-routes.ts` — POST /api/reflect

The exact prompt string sent to the language model is:

> "You are a placement engine for a platform called animologic. A person has just described their problem in their own words. Your job is to reflect back what you heard — plainly, accurately, without softening or reframing. Do not make it sound better than it is. Do not add encouragement. Do not interpret beyond what they said. If the problem is painful, say so plainly. If it is chaotic, say so plainly. If it is unclear, say so plainly. End with a single question: 'Do I have it right, or would you like to clarify?' Keep the reflection to 2–4 sentences maximum. No preamble. Start directly with the reflection."

The prompt is not configurable at runtime. It is a string literal in the source code. The model used is `claude-sonnet-4-6`.

Claim 04 — Server-Side Classification

Shape classification is always server-side. The client cannot assert a shape.

`server/animologic-routes.ts` — POST /api/place, functions `judgeShape()` and `classifyShape()`

The placement route calls `judgeShape(problem)` on the server. The client sends only the problem text. The shape returned by the server is what is stored. There is no code path that accepts a client-supplied shape without server classification.

`judgeShape()` first calls `classifyShapeLLM()`. If the language model is unavailable, it falls back to `classifyShape()`, a deterministic regex function. The six shapes and their regex patterns are defined in the `SHAPES` constant: rigidity, coordination, decay, coherence, trust, and scale. If neither method returns a confident classification, the route returns an explicit refusal rather than guessing.

Claim 05 — Anonymity Enforcement

The `assertAnonymous()` function throws an error if any identifying property is present in the data being written to the database.

`server/animologic-routes.ts` — function `assertAnonymous()`

The exact forbidden properties list in the code is: `name`, `email`, `phone`, `ip`, `ipAddress`, `userAgent`, `location`, `geo`, `deviceId`, `fingerprint`, `realName`. The function checks every key in the object being written against this list. If any match is found, it throws: "ANONYMITY VIOLATION — refused to write identifying properties onto a Handle". This throw propagates up and prevents the database write from completing.

`assertAnonymous()` is called in `dbPlace()` before every placement write. It runs on every write, not once at startup.

Claim 06 — Problem Text Storage

The platform stores the problem text as written, capped at 2,000 characters, attached to an anonymous handle.

`server/animologic-routes.ts` — POST /api/place, function `dbPlace()`

The route calls `String(problem).trim().slice(0, 2000)` before passing the text to `judgeShape()` and then to `dbPlace()`. The `dbPlace()` function writes `h.problem = $problem` to the Neo4j Handle node. The handle has no name, email, or identifying property.

This is a material disclosure. The problem text is what the person wrote. If they included identifying information in what they wrote, that information is stored. The platform does not strip or redact the text before storage.

Claim 07 — Contact Exchange

A reach is an offer and not a transfer. Nothing moves until both parties have consented independently.

`server/animologic-routes.ts` — POST /api/exchange/consent and POST /api/exchange/pass

POST `/api/exchange/consent` writes a `CONSENTED_TO_EXCHANGE` relationship from the requesting handle to the target handle in Neo4j. It also writes the contact detail to the relationship. Nothing is sent to the other party at this point.

POST `/api/exchange/pass` checks whether both a relationship from A to B and one from B to A exist. Only when both exist does it read both contact details and return them in a single response. After the pass, both relationships are deleted from Neo4j. The broker holds nothing after the exchange completes.

Claim 08 — Observation Fails Closed

Messages sent inside a collaboration space are observed. If the observation engine is unavailable, the message is not posted.

`server/animologic-routes.ts` — POST /api/space/:id/say

The route calls `judgeMessage(text)`. If `judgeMessage()` returns `ok: false`, the route returns HTTP 503 with the message: "This space is observed, and that check is unavailable right now. Your message was not posted. Nothing passes here unobserved — that is what you agreed to and what we owe you. Try again shortly." The message is not written to Neo4j. The system fails closed.

Claim 09 — Shared Audit Record

Both parties in a collaboration space see the identical audit record while both remain members. Withdrawal removes membership and with it access to the space audit.

`server/animologic-routes.ts` — GET /api/space/:id/audit, function `auditFor()`

`auditFor()` calls `isMember()` before returning any data. If the requesting handle is not a member of the space, it returns null, and the route returns HTTP 403. The `withdraw()` function deletes all relationships from the handle node, including `MEMBER_OF` relationships to spaces. After withdrawal, `isMember()` returns false, and the audit is inaccessible.

Both members call the same `auditFor()` function against the same Neo4j data. There is no second ledger and no owner-only view that differs from what members see.

Claim 10 — Device Key

The platform never stores the raw device key. In the event of a suspicious import attempt, a truncated hash of the key is logged. The hash cannot be reversed to recover the key.

`server/animologic-routes.ts` — POST /api/recover

The route never writes `deviceKey` to Neo4j. When a suspicious import is detected (same key imported twice within ten minutes), the route computes `crypto.createHash("sha256").update(deviceKey).digest("hex").slice(0, 16)` and writes that 16-character hex string to an Observation node. The raw key is not written. A SHA-256 hash truncated to 16 hex characters cannot be reversed to recover the original input.

Claim 11 — Security Headers

The platform sets four security headers on all /api/* routes: X-Content-Type-Options, X-Frame-Options, Referrer-Policy, and X-XSS-Protection.

`server/animologic-routes.ts` — security headers middleware

The middleware runs on every request whose path starts with `/api/`. It calls `res.setHeader()` for each of the four headers. `X-XSS-Protection: 0` is intentional: modern browsers ignore this header, and setting it to 1 can introduce vulnerabilities in older browsers. Setting it to 0 disables the legacy behavior.

Claim 12 — Rate Limiting and Client Hardening

Rate limiting is applied server-side. The ANIMO_LIVE object, the session key reference, and the placement threshold are frozen with Object.freeze in the browser.

`server/animologic-routes.ts` — function `limit()` • `client/public/client-patch.js` — Object.freeze calls

The `limit()` function implements an in-memory sliding window counter. The reflect route uses `limit(20, 60000)`. The placement and consent routes use `limit(30, 60000)`. The key is the session cookie value if present, falling back to the request IP address.

After the ANIMO_LIVE object is constructed and its methods are assigned, `Object.freeze(ANIMO_LIVE)` is called. The same is done for `ANIMO_KEY` and `__ANIMO_THRESHOLD__`. A frozen object cannot have properties added, removed, or changed. A person with console access cannot replace `ANIMO_LIVE.place` with a function that bypasses server classification.

Claim 13 — Domain-Aware Semantic Matching

Neighbors are matched not only by structural shape but by semantic domain. The domain is derived from the person's field text by the language model, not from a predefined taxonomy.

`server/animologic-routes.ts` — functions `classifyDomain()`, `scoreDomainSimilarity()`, `near()`

At placement time, `classifyDomain(field)` calls the language model and returns a 2–4 word canonical domain phrase (e.g., "intimate partnership", "technical leadership", "caregiving and grief"). This phrase is stored as `h.domain` on the Handle node. The field free-text is preserved unchanged alongside it.

When a placed person requests their neighbors via GET `/api/near`, the `near()` function fetches all same-shape candidates and their stored domain phrases. It then calls

`scoreDomainSimilarity()` once in a single batched LLM call, returning a semantic similarity score (0.0–1.0) for each candidate relative to the visitor's domain phrase.

Tier thresholds are stored in a `MatchingConfig` node in Neo4j and are operator-configurable via GET/PATCH `/api/operator/config`. The defaults are:

Tier	Condition	Default threshold
Tier 1	Similarity >= tier1Threshold	0.8
Tier 2	Similarity >= tier2Threshold	0.5
Tier 3	Similarity < tier2Threshold	< 0.5

If either side has no stored domain phrase (handles placed before this feature shipped), the system falls back to field string equality for tier assignment.

Claim 14 — Composite Resonance Scoring

Within each tier, neighbors are ranked by a composite resonance score derived from three signals: posture, velocity, and orientation. The weights follow the golden ratio.

`server/animologic-routes.ts` — functions `scoreResonance()`, `computeResonanceScore()`, `resonanceLabel()`

At placement time, `dbPlace()` also stores a 300-character `h.seamSummary` derived from the person's problem text. This summary is used for resonance scoring without exposing the full problem text to neighbors.

When `near()` assembles the neighbor list, it calls `scoreResonance()` with the visitor's seam summary and the summaries of all candidates. A single batched LLM call returns three sub-scores per candidate:

Signal	What it measures	Weight
Posture	How well their stances toward the problem complement each other (inside/through, resisting/emerging, etc.)	0.382
Velocity	How well their urgency levels match (acute/acute, chronic/chronic, acute/chronic)	0.382
Orientation	Whether their sense-making vs action-seeking orientations are compatible	0.236

The weights 0.382, 0.382, 0.236 sum to 1.0 and follow the golden ratio decomposition (phi^-2 + phi^-2 + phi^-3 ≈ 1). The composite score is `computeResonanceScore(posture, velocity, orientation)`. Neighbors within each tier are sorted by this score descending. The score is not shown as a number in the UI; it is expressed as a label: "strong resonance" (>= 0.7), "moderate resonance" (>= 0.4), or "low resonance" (< 0.4).

Claim 15 — Seam Classification Audit

Every classification decision at the seam is persisted as a SeamEvent node in Neo4j. The operator can review accepted and rejected entries, flag false positives and false negatives, and override the stored shape.

`server/animologic-routes.ts` — function `createSeamEvent()`, routes GET/PATCH `/api/operator/seam-events`

A `SeamEvent` node is created at every classification decision point: successful placement, unknown-shape rejection, rate-limit rejection, shape-mismatch rejection during key import, and successful key-import recovery. Each node stores: `id`, `at` (timestamp), `handleId` (if known), `rawText` (first 500 characters of the problem text), `classifiedShape`, `confidence`, `engine` (llm or regex), `outcome` (accepted/rejected/unknown/rate-limited/shape-mismatch), `reason`, `overrideShape` (null until operator overrides), `flaggedFP` (false positive flag), and `flaggedFN` (false negative flag).

The operator portal Seam Review tab fetches all events via GET `/api/operator/seam-events` (owner-only). The operator can filter by outcome, flag any event as a false positive or false negative, and override the stored shape via PATCH `/api/operator/seam-events/:id`. Override and flag writes are also persisted to the SeamEvent node.

Claim 16 — Notification Tier Filter

When subscribing to neighbor notifications, a person can choose whether to receive alerts for all same-shape neighbors or only those in the same semantic domain.

`server/animologic-routes.ts` — POST /api/notify/subscribe, function `fireNeighborNotifications()`

The `NotificationPreference` node stores a `tierFilter` field with two possible values: `all` (notify on any same-shape placement) or `domain` (notify only when the new arrival's domain similarity score meets the tier 1 threshold). The subscribe form in the placed pane presents this as a radio choice: "All neighbors in my shape" or "Same domain only".

`fireNeighborNotifications()` reads `tierFilter` for each subscriber and skips the notification if the new arrival's domain similarity falls below the tier 1 threshold when `tierFilter = domain`.

Claim 17 — Key Import Shape-Check Confidence Gate

The shape-check during device key import uses a confidence threshold. A high-confidence mismatch refuses the import. A low-confidence result passes with a warning.

`server/animologic-routes.ts` — POST /api/recover

The route calls `judgeShape(description)` on the description the person provides during import. It then compares the classified shape against the shape stored on the handle associated with the key. If the classified shape differs from the stored shape and the confidence is above 0.7, the import is refused and the key is cleared from localStorage. If the confidence is below 0.7, the import proceeds with a warning message: "We couldn't confidently match your description to your placement. Proceeding, but please verify your placement is correct."

Claim 18 — Human-Only Post-Seam Interaction

All post-seam interaction is between humans only. The AI serves as infrastructure and classification engine. No AI agent participates in any conversation, exchange, or collaboration space.

`client/index.html` — consent section (terms v1.7)

Terms version 1.7 includes a clause stating that all post-seam interaction is between humans only, that the AI serves as infrastructure and classification engine, and that no AI agent participates in any conversation, exchange, or collaboration space. This clause is rendered in the consent section and must be accepted before crossing the seam.

> Design principle: The queen is not a god and does not attempt to become one. She is not a figure of authority or a stand-in for the divine. She is a presence that reminds the hive that something larger is at play in each person's life, whatever they understand that to be. The structural connection between every placed handle and the queen exists so that reminder is always present in the data, not so that she can direct what happens inside the hive.

Claim 19 — No Browser Prompt Dialogs

No user-facing action on the platform uses window.prompt(), window.alert(), or window.confirm(). All input collection uses inline HTML forms.

`client/index.html` — `#key-import-check-form`, `#recover-key-check-form`, `#commons-report-form` • `client/public/client-patch.js` — `showKeyImportForm()`, `hideKeyImportForm()`, `wireLandingKeyImport()`

Three former `window.prompt()` calls have been replaced with inline HTML forms that render inside the page without triggering browser-native dialogs. Browser-native prompt dialogs are blocked in many mobile browsers, PWA contexts, and iframe environments. They also cannot be styled, cannot be dismissed without losing the input, and do not work with assistive technology.

Former prompt	Replacement element	Location
Key import shape-check	`#key-import-check-form`	Placed pane, inside the environment
Landing-page key import	`#recover-key-check-form`	Recovery section, before the seam
Commons report text	`#commons-report-form`	Commons section, inside the environment

Each form is hidden by default and shown inline when the triggering action fires. Each has a cancel path that hides the form and restores the prior state without a page reload.

Claim 20 — Three Documented Re-entry Paths

Three distinct re-entry paths exist. The seam is required on every new device. The device key file is a continuity token, not a bypass.

`client/public/client-patch.js` — `boot()`, `showKeyImportForm()`, `wireLandingKeyImport()`, `wireRecoveryForm()` • `server/animologic-routes.ts` — POST /api/recover, POST /api/recover-by-shape

Path	Seam on this device	Code
Same device, key in localStorage	No — already crossed	`boot()` reads `animo_device_key`, calls `isCurrent()`, fires `animo:consented` automatically
New device, key file available	Yes — seam crossed first or concurrently	`showKeyImportForm()` or `wireLandingKeyImport()` → POST `/api/recover` with shape-check description

Path	Seam on this device	Code
New device, no key file	Yes — seam crossed first	`wireRecoveryForm()` → POST `/api/recover-by-shape` semantic match against stored seam summaries

The device key file is not a bypass. It is a continuity token. Arriving on a new device with a key file does not skip the seam. The seam is crossed on the new device before or during the import. The key file provides the shorthand: the person arrives already knowing who they are here, and the shape-check description confirms that the key belongs to them.

The shape-check on import requires the person to briefly describe the problem they brought here. The server compares this description against the stored seam summary using the same LLM classification pipeline. A high-confidence shape mismatch refuses the import and clears the key from localStorage. A rate check also prevents the same key from being imported twice within ten minutes.

What This Briefing Does Not Claim

This briefing does not claim that the platform has been independently audited, certified, or reviewed by any third party. It has not. This briefing does not claim that the language model is always available or that its outputs are always accurate. It is not and they are not. This briefing does not claim that the regex fallback classifier is exhaustive. It covers six shapes with specific vocabulary. Problems described without that vocabulary may not classify.

The domain classification and resonance scoring systems depend on language model availability. When the model is unavailable, domain classification is skipped and tier assignment falls back to field string equality. Resonance scoring is skipped and neighbors are returned in placement order within each tier.

Every behavioral claim in this briefing corresponds to a specific, deployed, verifiable location in the production codebase. The codebase is the authority. This document is a guide to it.