EquipCert AI — Comprehensive Technical Overview

AI-Assisted Equipment Inspection, Evidence Capture, and Compliance Operations

A client-readable and engineer-ready reference for inspection workflows, equipment records, photo and signature evidence, offline behavior, AI assistance, interface design, security engineering, US and EU legal posture, audit accountability, horizontal scalability from ten thousand toward one million concurrent users, availability engineering, verification, operations, a phased roadmap, and maintainable developer handoff.

Next.js + Flutter | Supabase | Contentful | Multi-Provider AI | Docker + Kubernetes-Ready

Source-Reconciled Public Technical Edition | Early-October 2026

Developed by Zahidul Islam

Review on 14 October 2026 — latest verified evidence

This review adds current evidence to the existing technical overview. Every earlier passage, dated update, checklist and native table is retained below. Read this note first for the latest checks; the early-October and September editions remain the historical record. The compact layout implements the requested 10–15-page edition without removing previous material.

Current service status

The public deployment answered successfully in all seven existing production checks on 14 October: HTML delivery, liveness, readiness, the nine expected security headers, analysis endpoint mounting, the intended backend in the shipped JavaScript, and backend reachability. The three dependency checks also passed: authentication, database REST reachability and checklist-content delivery. These probes establish the specific responses tested; they do not constitute an authenticated inspection walkthrough or verify every account workflow.

Current code and automated evidence

The local checked-out commit remains the October 1 release, 5cd2a96. The comparison of tracked working files with that checkout found no substantive difference after line-ending differences were ignored. This review documents renewed verification, rather than a newly implemented feature release.
The web unit suite was rerun: 104 tests passed, zero failed and zero skipped, across 33 suites. The configuration-boundary check passed across 157 files and nine rules with zero violations. Earlier counts in the retained edition describe their original runs and should be read with their dates.

Scheduled operations

Public GitHub Actions records show the automated production check succeeded on 14 October, including run 38549097707, and the database keepalive succeeded on 13 October, run 38408671502. These observations support that the restored service and its maintenance automation remain active at the time checked. They do not establish a continuous-availability percentage or prove that every scheduled interval executed.

Remaining boundaries

The previous limitations still matter. This review does not close the missing web component/browser-flow suite, persist electronic-signature consent as evidence, configure commercial payments, prove an off-machine signing-key backup, resolve test-fixture isolation, or provide a completed restore drill. Credential-rotation completion is not established by the checks performed here.
The 429 native-client tests and successful Android/iOS build evidence below remain dated October 1 evidence; the mobile suite, full builds, lint and security scans were not rerun for this document-only update. No paid AI inference, payment transaction, deployment or database migration was performed. Large-user capacity figures, service-level objectives and future infrastructure tiers remain design targets unless their own sections explicitly give measured evidence. Legal and regulatory passages are preserved as previously documented. This review does not represent a fresh legal assessment, counsel approval or certification. Existing implementation gaps and historical incident descriptions remain visible.

Evidence references

Production check: https://github.com/Zahidulislam2222/equip-cert/actions/runs/38549097707
Database keepalive: https://github.com/Zahidulislam2222/equip-cert/actions/runs/38408671502

Earlier editions and complete technical reference follow

Current Evidence and Scope Note — Early-October 2026

This edition extends the late-September edition. Everything that edition recorded is kept below, including its status note; where a statement has changed since, it is corrected in place with the date 1 October 2026.

Status on 1 October 2026 — read this first

The live site works end to end again. The hosted database that was unreachable on 24 September is reachable, the live site has been rebuilt from the latest code and redeployed, and every automated check that was failing because of the outage now passes — including, for the first time since 11 September, every job in continuous integration.

The outage had a precise cause, now fixed. The database runs on a free hosting tier that pauses a project after a week without database activity. A scheduled keep-alive job existed and reported success twice a week — but it called a health endpoint that never touches the database, so to the provider the project looked idle. The keep-alive now runs a real database query every day and only reports success when a row comes back; its first run against the restored project returned a row.

What changed since the late-September edition

- Audit log live. The database-written, erasure-compatible audit log described in section 14 was applied to the hosted database after both safety pre-checks were re-run against it. The full tenant-isolation and audit suite then passed 28 of 28 against the hosted database, including the three erasure cases that had been the reason for the design.
- A payment defect fixed before it could fire. On the serverless hosting target, the subscription webhook would have rejected every genuine payment event: the platform parses the request body before the handler sees it, and the payment provider signs the exact original bytes. The widely published fix for this applies to a different framework and would have changed nothing here. The handler now reads the exact bytes on both deployment targets without triggering the platform's parser, and its log no longer records event contents, which can include customer details. Proven with provider-signed test payloads over a real HTTP server: run against the previous handler, the same tests fail exactly the platform-replay and raw-stream cases; the new handler passes all of them. On the self-hosted target the full request path was exercised; for the serverless target the behaviour was verified against the platform runtime's own source code, not on the hosted platform itself.
- Example credentials can no longer be mistaken for real ones. A credential that still holds the shape of the example configuration file is now treated as not configured, so the service says so instead of failing with a misleading error. Only the setting's name is logged, never a value.
- The disclosure check now exists. The late-September edition noted that nothing compared the browser's content security policy with the published list of third parties. The build now fails if the policy allows any external address that the list does not name. Seven deliberately planted violations — including bare hosts, wildcards, upper-case schemes and a whole-scheme allowance — were each caught.
- Explicit signing consent on mobile. The mobile client now asks for the same explicit "I agree to sign electronically" consent as the web client before the signature pad accepts a stroke, and consent cannot be withdrawn while a signed inspection is being filed. Withdrawing it beforehand clears the signature.
- More tests. Web unit tests rose from 43 to 104, now covering the payment webhook, configuration handling, the breached-password check (including that only a five-character hash fragment ever leaves the device), and plan gating. Native client tests rose from 423 to 429.
- Narrower automation credentials. The credential continuous integration uses for the hosted database is now read-only, limited to this one project, and expires on a fixed date; the database password is supplied only to the steps that need it.

- Two honest findings. First, both clients now ask for signing consent, but neither yet stores that consent as a separate piece of evidence, so it could not be produced later on its own; this is on the roadmap. Second, the live test suite's clean-up step had been silently ignoring its own failures since early September. Part of that was the erasure defect fixed by the audit-log work; the rest is by design, because the suite signs inspection records to prove they are immutable, and an immutable record cannot then be deleted. The clean-up now reports every record it cannot remove, and the recommended next step is a separate test environment.
- Monitoring cadence measured, not assumed. The production check is scheduled every thirty minutes, but the record of its runs shows the hosting service actually started it every two to five hours. Outages are therefore detected within hours, and that is the figure this document now uses.

Verified snapshot — 1 October 2026

- Continuous integration on the latest commit: all four jobs pass — web static gates, Android, iOS, and live services against the hosted database.
- Live deployment rebuilt from the latest commit: 144 of 144 deployed files byte-identical to the built release by SHA-256 checksum, the reverse-proxy configuration identical to its generated source, and five public pages byte-identical to the local build when fetched through the content-delivery network.
- 7 of 7 automated production checks passing against the redeployed site.
- 28 of 28 tenant-isolation and audit tests passing against the hosted database; all eight migrations applied and tracked.
- 104 of 104 web unit tests on Windows and on a clean Linux container matching the build runner; 429 native client tests passing; native static analysis reports no issues.
- Configuration-boundary enforcement across 157 files, nine rules, zero violations.
- Database vendor's security advisor: one accepted warning, nothing blocking; performance advisor: zero findings.
- Independent fresh-context review of the change set: ready with fixes, zero blockers; the fixes were applied and every gate re-run.

Evidence and Scope Note — Late-September 2026 (kept for the record)

This edition extends the mid-September edition, which superseded the early-September edition. Everything the mid-September edition recorded is kept below; where a statement has since changed, it is corrected in place with its date, and the newest work is summarised first. The early-September edition recorded a system that had become verifiable: a live backend, continuous integration, and authored tests. The mid-September edition recorded the next stage of work, which was deliberately aimed at the three questions a serious client or engineering reviewer asks before trusting a product with real operations: Is it lawful? Is it secure? Will it hold up at scale?

Each of those questions was answered with built, tested engineering rather than with adjectives, and each answer is graded honestly. Where something is designed but not yet measured, it is labelled as a target on a named deployment tier, never as an achieved result.

Status on 24 September 2026 — resolved by 1 October 2026

The live site's pages, security headers, and health endpoints are still served, but the hosted database project the live build was compiled against is no longer reachable — its address no longer resolves. Sign-in on the live site therefore fails until the database is restored and the site is rebuilt against it. This is disclosed rather than hidden: the continuous-integration job that tests live services fails on it, the public repository states it, and the automated production check in section 22 exists to report exactly this condition. Every figure below described as verified against the live deployment was measured on 15 September 2026, before the outage, and is dated where it appears. *Update, 1 October 2026: resolved — the database is reachable and the site has been rebuilt and redeployed; see the early-October note above.*

What changed since the mid-September edition

- Public documentation. The source repository now publishes a documentation set derived from the engineering records: a legal and regulatory map across twelve US and EU regimes, an architecture document with its decisions and the reasons for them, a threat model organised by the STRIDE categories, a roadmap to one million concurrent users with measurable exit criteria per phase, an accessibility statement, a contributing guide, a changelog, and a licence. Each uses the same evidence labels as this document.
- Wider legal coverage. The legal map adds the EU Cyber Resilience Act, the NIS2 Directive, the European Accessibility Act, both relevant titles of the Americans with Disabilities Act, US state privacy laws and Global Privacy Control, and the current state of EU–US data transfers. Section 17 summarises each.
- Two undisclosed third parties found and disclosed. Writing the threat model from the deployed content security policy revealed two public services the sub-processor list did not name: a reverse-geocoding service that receives inspection coordinates, and a breached-password check that receives a five-character fragment of a password hash. Neither offers a processing agreement. Both are now listed separately in the public sub-processor list and the record of processing; the privacy policy's text still has to be updated to name them.
- Claims corrected. An independent review of the new documentation returned *not ready* with six major findings, all fixed before anything was published. Several corrections apply to earlier editions of this document as well and are made in place below: row-level security is forced on ten of eleven tables, not every table; the audit log is built and verified but not yet applied to the hosted database; the security workflow runs secret detection and static analysis, while dependency updates arrive as automated update requests rather than through a separate dependency-review job; and explicit consent to sign electronically exists on the web client, while the mobile client currently shows a notice.
- Private vulnerability reporting enabled. The published security policy pointed researchers to the repository's private reporting channel, which turned out to be switched off. It is now on.
- A cross-platform test defect. The first build after publication showed the web unit suite failing all 43 tests on the Linux build runner while passing 43 of 43 on the development machine. The tests invoked a bundler in a way that works only on Windows. They now use the bundler's programming interface and pass 43 of 43 on both.
- Dependencies. Five automated update requests — three for continuous-integration actions and two grouped package updates — were rebased, passed every job, and were merged. Two major-version updates are held until the web client has tests able to catch their failure modes, and one major version is declined until the lint tooling supports it.

What changed since the early-September edition

- Legal. Versioned Terms of Service written against US and EU electronic-signature law and the relevant workplace-safety inspection rule; a rewritten privacy policy; machine-readable AI-content marking embedded in every generated inspection report to meet the EU AI Act's transparency duty for AI-generated content.
- Accountability. An audit log that previously existed but received no entries is now written automatically by the database itself for every material change, cannot be forged or edited by any application user, and still honors the GDPR right to erasure — two requirements that conflict directly and were reconciled in one design.
- Security. Rate limiting that stays correct across many server instances, a published vulnerability-disclosure policy with safe harbour, an industry-standard security contact file, additional browser isolation headers, request timeouts that are actually enforced, and authentication settings managed as code with drift detection.
- Scalability. The origin now scales horizontally by configuration: any number of replicas behind a health-checked load-balanced pool on a single host, or a generated Kubernetes deployment that autoscales from three to fifty instances with disruption budgets and network isolation. A written capacity model covers ten thousand, one hundred thousand, and one million concurrent users, backed by a load-test plan with pass/fail thresholds.
- Availability. Separate liveness and readiness probes, a graceful drain sequence verified in the production container image, published service-level objectives per deployment tier with error budgets, recovery objectives, failure-mode analysis, and step-by-step runbooks.
- Production monitoring. A automated production check that asserts the live site is not merely up but actually functional — including that it is wired to the correct backend. It was built because the site was once found returning success on every page while nobody could sign in.
- Deployment. The live deployment was rebuilt onto the current backend and the current code, with file-by-file checksum parity proven between the built artifact and the served one, and automated rollback protection on the reverse-proxy change.

Verified snapshot

- Configuration-boundary enforcement across 139 files, eight rules, zero violations.
- 43 of 43 unit tests passing, covering consent, rate limiting across replicas, store failure behavior, and report provenance marking. *104 as of 1 October.*
- 28 of 28 database tests passing on a clean local database built from migrations, covering tenant isolation and the new audit-log behavior, including erasure.
- 423 passing automated tests for the native client, unchanged and still run on every change. *429 as of 1 October.*
- 13 of 13 server-lifecycle checks passing inside the production container image: readiness flip, drain, clean exit, and slow-client cut-off.
- 143 of 143 deployed files byte-identical to the locally built release, proven by SHA-256 checksums.
- 7 of 7 automated production checks passing against the live deployment on 15 September 2026: site, liveness, readiness, nine security headers identical to their single source, API mounted, correct backend compiled in, backend reachable. *As of 24 September the backend is unreachable — see the status note above. 1 October: reachable again; 7 of 7 against the redeployed site.*
- Measured static-tier throughput of approximately 3,600 requests per second per CPU core with zero failures at 300 concurrent connections.
- 43 of 43 web unit tests passing on Windows and on a clean Linux container matching the build runner (24 September).
- Continuous integration on the latest commit: web static gates, Android, iOS, and security scanning pass; the live-services job fails because the hosted database is unreachable. *1 October: all four jobs pass, including live services.*
- Zero known vulnerabilities in production dependencies after the September dependency updates.

1. Reading Guide and Evidence Language

This document is written for two audiences. Client readers can use the product, legal, security, scalability, availability, status, and review sections to understand what the system does, what protects them, how far it can grow, and which operational decisions remain theirs. Developers can use the architecture, module, data, trust-boundary, failure, test, and handoff sections as a source-reconciled implementation map. The public source repository carries a documentation set derived from the same records — a legal map, architecture, threat model, roadmap, and accessibility statement — and uses the same evidence labels.

Four evidence words have distinct meanings throughout:

Word	Meaning
Measured	A command was run and the number came out of it. The command can be re-run.
Verified	An automated check exercised the behavior for real, and its failure would have been visible.
Built	The code or configuration exists and has been validated, but has not been exercised at the scale or in the environment it is for.
Designed for	What the architecture is built to reach on a named deployment tier. Not a measurement and not a promise.

A note on numbers. Every figure in this document has a command behind it. Where a capability has been designed but not measured, it is described as a path with its requirements. That distinction is kept deliberately: a single unverifiable number invites a reviewer to discount everything around it.

2. Executive Summary

EquipCert is a tenant-aware equipment inspection application. A technician selects or registers equipment, captures identifying and condition evidence, answers a checklist, records location and a signature, and submits an inspection. Managers receive dashboard, inspection, equipment, corrective-action, scheduling, and notification views. A client-side renderer turns a stored inspection into a shareable PDF report that carries machine-readable provenance.

The browser application is a statically exported Next.js client. The native application is a separate Flutter client for Android and iOS that talks to the same backend and enforces the same schema and disclosure contracts. Supabase supplies authentication, relational persistence, row-level rules, private storage, and realtime events. Contentful supplies checklist content. Server-side handlers perform AI image analysis through one of three interchangeable providers, receive payment events, and receive data-subject requests.

The latest stage of work turned three client questions into engineering: legal compliance across US and EU frameworks as far as software can carry it, layered security with verified controls, and an architecture whose path from today's single container to hundreds of thousands of concurrent users is configuration and provisioning rather than a rewrite. What remains unproven is stated plainly in the risk register: the database tier has not been load tested, multi-replica availability has not yet run for a measurement period, the new audit-log migration awaits application to the live database, payment processing is unconfigured, and no independent legal or security review has taken place. Since then, the hosted database the live site depends on has become unreachable; restoring it is the first item on the roadmap. The path forward is published as a phased roadmap in which each phase ends with measured exit criteria rather than a date.

3. Product Purpose and Boundaries

Intended value

- Reduce paper and re-entry in field inspections.
- Present a repeatable checklist matched to an equipment type.
- Preserve technician, time, result, location, media, and signing context.
- Surface failed inspections for corrective handling.
- Support weak-connectivity work through a local queue.
- Expose organization-scoped management views with an accountable change history.
- Serve web and native users from one backend, one schema, and one set of compliance rules.

Explicit boundaries

- AI output is a suggestion and cannot determine identity, safety, or compliance. The application records that a value was machine-suggested, separately from the technician's confirmation of it.
- The PDF reflects application data; it is not independent certification.
- An affirmative checkbox and signature interface support electronic-signature law but do not by themselves prove every legal requirement for a particular transaction.
- Client route guards and hidden controls are not authorization boundaries. Authorization is enforced in the database.
- Legal and safety-standard alignment is engineering and product design intent, not regulatory approval or legal advice.
- Team and Reports remain placeholder screens. Schedule and Settings are limited views rather than full administration systems.

4. Verified Implementation Snapshot

Area	State	Evidence grade
Web client routes	Fourteen authored page modules plus Terms, Privacy, and request intake	Verified by build
Native client	Flutter, Android and iOS, eight screens	Verified by build and 423 tests; 429 as of 1 October
Database schema	Eight versioned migrations	Seven verified live; the audit-log migration verified locally, awaiting live application. *1 October: all eight applied live and tracked*

Area	State	Evidence grade
Authorization	Row-level security enabled on all eleven tables and forced on the ten holding tenant data; privilege-guard triggers	Verified by authorization suites
Audit accountability	Database-written, append-only, erasure-compatible audit log	Verified locally, 28 of 28; *live since 1 October, 28 of 28 against the hosted database*
Evidence storage	Private bucket, organization-scoped object paths	Verified by policy and by test
AI assistance	Three interchangeable providers behind one fail-closed handler	Implemented; provider contract not exercised
AI-content marking	Machine-readable marking embedded in generated PDFs	Verified by rendering a real PDF
Legal documents	Versioned Terms of Service, privacy policy, five compliance positions; public legal map across twelve US and EU regimes	Implemented; not counsel-reviewed
Security disclosure	Vulnerability policy, safe harbour, standard security contact file, private reporting channel enabled	Implemented and live
Rate limiting	Correct across replicas via a shared store, safe fallback	Verified by unit tests
Horizontal scaling	Replica pool on one host; generated Kubernetes deployment 3–50	Built and validated; not load tested
Server lifecycle	Liveness, readiness, graceful drain, enforced timeouts	Verified in production image, 13 of 13
Production monitoring	Automated functional check of the live deployment	Verified against live, 7 of 7 on 15 September; built to report the database outage found on 24 September. *7 of 7 again on 1 October*
Capacity	Static tier measured; database tier not load tested	Partially verified
Continuous integration	Four jobs plus security scanning	Verified; the live-services job fails while the hosted database is unreachable. *1 October: all four jobs pass*
Mobile distribution	Neither platform published to a store	Not verified
Hosted database behind the live site	Unreachable as of 24 September 2026; *reachable again by 1 October*	Verified by name resolution and the live-services job; *1 October: verified by 7 of 7 production checks and the passing live-services job*
Electronic-signature consent	Explicit control on web; notice on mobile. *1 October: explicit control on both; consent not yet stored as separate evidence*	Partial
Public documentation	Legal map, architecture, threat model, roadmap, accessibility statement	Published with the source
Payment webhook signature handling	Exact-bytes verification on both deployment targets (added 1 October)	Verified with provider-signed payloads; self-hosted path end to end
Disclosure check	Build fails if the content security policy allows an unlisted third party (added 1 October)	Verified by seven planted violations

5. Architecture Overview

The system has four cooperating parts and two delivery targets, and every architectural choice below was made so that growth is a configuration change.

The web client is a statically exported Next.js application. It has no server components, no middleware, and no framework-level API routes; every page is a static asset and all routing and session handling happens in the browser. This keeps the hosting surface trivial, lets the entire site be served from a content delivery network edge, and means the same build artifact can be served by an edge network or by a small self-hosted process without behavioral divergence. It is the first of the three properties the scaling story depends on.

The native client is a Flutter application for Android and iOS. It shares the backend, schema contract, and disclosure requirements with the web client, and its test suite asserts those shared contracts directly against the other client's source. A drift between the two clients fails a test rather than surfacing in the field.

The backend is Supabase: authentication, relational persistence, row-level security, private object storage, and realtime events, provisioned in an EU region. Every tenant-scoped table carries an organization identifier and a policy set. Signed inspection records are immutable by policy, and every material change is recorded in an audit log written by the database itself — once the audit-log migration is applied to the hosted database (section 14).

The server handlers are standalone functions for AI analysis, payment events, and data-subject requests. They are the only components permitted to hold server-only credentials.

Deployment shape

Two targets are maintained from one source. The first is static hosting with edge-delivered security headers and serverless handlers. The second is a self-hosted container behind a reverse proxy and content delivery network, in which a small Node adapter serves the exported assets and mounts the same handler code unmodified.

The self-hosted target is generated, not hand-written, from one deployment configuration file. That file declares the replica count, resource limits, health-check cadence, drain timings, and connection timeouts. From it a generator produces the reverse-proxy site definition, the container composition, and a complete Kubernetes deployment, and it refuses to generate a configuration whose timings would cause dropped requests — for example a proxy keep-alive longer than the server's, or an orchestrator stop grace shorter than the drain. Continuous integration fails if the committed Kubernetes manifests drift from that configuration.

The container is built from a copy-only definition on a digest-pinned base image and runs as an unprivileged user, with a read-only filesystem and all Linux capabilities dropped in the orchestrated form.

6. Client Surfaces and Route Ownership

Surface	Routes	Access
Marketing	Landing, Pricing	Public
Legal	Terms of Service, Privacy Policy	Public
Privacy operations	Data-subject request intake	Public
Security	Standard security contact file	Public
Authentication	Login, Signup	Public

Surface	Routes	Access
Application shell	Dashboard, Inspections, Equipment, Schedule, Team, Reports, Settings	Authenticated
Technician flow	Inspect	Authenticated
Operations	Liveness and readiness probes	Infrastructure

Route guards in the application shell are a usability affordance. They decide what a signed-out visitor sees; they do not decide what data a request can reach. That decision belongs entirely to the database.

7. How the Web Interface Was Built

The interface is a deliberate engineering artifact, not a template, and the reasoning behind it is part of what is handed over.

Design system

The shipping design is dark-first: warm near-black surfaces, a single high-visibility Safety Yellow accent, and a display and body typeface chosen for this domain. Surfaces are warm neutral rather than blue-grey, because blue-grey reads as generic dark-mode software.

A single stylesheet is the sole owner of every design token. The native client reads the same tokens, and a test in the native suite parses the web client's stylesheet and fails if the two diverge. Three semantic colors — success, warning, and destructive — are reserved for inspection state and never used as decoration. Their foreground pairings are deliberately not uniform, because the uniform choice failed contrast requirements on the pass and fail controls; a test asserts them.

Design direction: "night shift"

The design direction was agreed with the owner before any code was written, with thirty-six acceptance criteria: deepen a concrete-and-shadow industrial language rather than adopt a generic software look, apply it to every public page and every application screen, and give the site one signature interaction. The work was then pushed further after the owner rejected an early pass as lacking feeling, and research into award-level scroll experiences changed the diagnosis. That research showed that unrelated brands become visually indistinguishable through the same structural habits — section counters, visible sequence markers, repeated eyebrows, identical copy alignment in every section — and that the page was doing several of them. All were removed: counters deleted, eyebrows cut from four to one, and copy alignment varied so no two neighbouring sections sit alike.

The signature interaction: an inspection you scroll through

The landing page leads with an interactive, real-time three-dimensional fire extinguisher. As the visitor scrolls, the camera walks the equipment the way an inspector would — charge gauge, operating mechanism, discharge path, result — and the copy for each point appears beside it while numbered markers track the moving geometry. The visitor can also take the camera: drag or use the arrow keys to rotate the object, select a marker to fly to that part, or separate the parts into an exploded view and reassemble it.

- Real geometry, one source. The model is exported from the same Blender scene definition that rendered the earlier film — twelve named parts, 18,232 triangles, a 444 KB file — so the object on the page is provably the product's own geometry rather than a lookalike. One content file owns the camera path, hotspots, explode offsets, light rig, and copy, so a label cannot name a part the camera is not looking at. Camera stops are specified as a feature point plus azimuth, elevation, and distance, never raw coordinates.
- Journey, not slideshow. The camera *lingers* at each inspection point while the copy peaks, then accelerates toward the next — the quality the owner described as "journey feeling". Scroll velocity feeds inertia: the object leans and leads its rotation slightly when scrolled fast, with a faint exposure lift like a real lens moving through light. Measured across 48 sampled frames, the motion rhythm between travel and settle rose from 4.7x to 24.3x with zero static frames.
- A world, not a floating object. The scene is lit like a set: a raking key light from above-left and an accent light behind the object, fog, and a ground plane carrying the page's grid that fades into the dark. A post-processing chain adds bloom, a vignette, and depth of field whose focus is re-aimed every frame at the part being inspected, so the inspected component is sharp and the rest falls away. Deterministic surface-detail maps replace the procedural texture the file format cannot carry.
- The page performs its headline. A blank inspection record docked along the bottom of the hero fills itself in as the camera walks the equipment, and is signed at the end. The product's promise is shown before a word of it is read.
- Lighting was a measurement. The first light rig placed the key almost on the camera axis and the model read as flat product imagery; with the key behind the object, every surface facing the viewer was in shadow and the red body read brown. Both were diagnosed geometrically and corrected against the real asset.

The 3-D runtime is code-split into its own chunk that the landing page's HTML does not reference, so phones, touch devices, and reduced-motion visitors download neither the engine nor the model and receive a complete still-and-list composition instead.

Motion system and feel

- Preloader as the first inspection. The opening moment is the extinguisher's own pressure gauge coming up to charge — the needle sweeps off its peg and settles in the green band while registration ticks lock on. It plays at most once per session when preference consent allows remembering, is skipped entirely under reduced motion, never exists without scripting, and has a hard ceiling of 3.2 seconds, so a stalled asset can delay the reveal but never prevent it.
- A reticle cursor — a registration mark echoing inspection equipment — that follows the pointer without ever hiding the system cursor.
- Magnetic calls to action that pull toward the pointer, applied to an element *inside* the link so the clickable area never moves away from the user.
- Animated cards with capped tilt, staggered grid entrances, animated counters, drawn lines, clip-path text reveals that never split words into per-character fragments (which would break screen readers), and bounded layered parallax that moves only decoration.
- Sticky stacking cards for the step-by-step product story, a bento grid in which offline-first work takes the widest tile because it is the capability the product is actually bought for, and disclosure cards that answer "enforced where?" for each technical claim.
- Scroll-driven animation everywhere. One major browser still ships native scroll-driven animation behind a flag, so roughly one visitor in six was seeing a completely static page. A roughly 700-byte inline script now detects support and falls back to intersection observation, with no hydration flash, nothing hidden when scripting is off, and a watchdog that reveals anything that never enters view.
- Authentication pages carry a purpose-rendered portrait still of the scene, 12 KB, replacing an abstract blurred shape that animated forever and ignored reduced-motion preferences.

Earlier rendered sequence — two measurements worth keeping

Before the real-time scene, the hero was a scroll-scrubbed film rendered in a physically based renderer. Two measurements changed it materially and remain standing checks: 53 percent of frames were visually static until a slow drift was carried through each dwell, bringing it to zero; and body copy composited over the real frames measured 4.17:1, below the 4.5:1 minimum, until the scrim was raised to 5.71:1. AI video generation was evaluated and rejected on evidence: the available models regenerate a camera path between two frames rather than following an authored one, which cannot keep a label attached to a known part. Generated-media spend for the entire interface: zero.

Loading-sequence correctness

A visitor reported that the hero looked broken for a few seconds on every reload and then corrected itself. The cause was verified on the live site: the numbered markers are positioned only by the 3-D render loop, so until the model finished loading — roughly two to three seconds — all three sat stacked in the corner of the hero, over the headline. They now remain hidden until the scene reports it is ready. The fix changed behavior only; the approved visual design was left untouched. The lesson is recorded in the codebase: anything positioned by a render loop needs an explicit not-yet-positioned state, because the first frame of the page is not the first frame of the loop.

Accessibility, treated as a requirement

The copy for every beat lives in a semantic ordered list in reading order, always present in the markup. Visitors on phones, visitors who have asked for reduced motion, screen-reader users, and anyone whose 3-D scene fails to load read the same list as ordinary content. The list is the truth; the animation is the presentation. A missing main landmark, a missing skip link, and six undersized navigation targets were found and fixed.

One rule learned expensively is non-negotiable: the critical path is never gated behind an entrance animation. The sign-in form once rendered inside three nested transparent animation wrappers, so if the animation did not complete nobody could sign in. Forms and primary actions now render visible from the markup.

What is deliberately fenced

The visual design is closed to change by the owner's instruction, with one authorized exception: a contrast correction that raised six failing token pairings, including white on the dark pass control at 2.922:1 — the worst ratio in the project, on a safety-critical button. Only lightness and three foreground pairings changed.

8. Roles, Identity, and Authority

Identity comes from the authentication provider. Organization membership and role live in a profile row. Three roles are recognized: administrator, manager, and technician.

Role changes are rejected at the database boundary by a trigger. Only an administrator may change a role, and an administrator may not change their own — self-promotion and accidental self-demotion are both refused. Role and organization are resolved through database functions that read the authenticated session, never from a value supplied by the caller. A client that lies about its role changes nothing. Every role change is now also written to the audit log with who made it and whom it affected.

9. End-to-End Flows

Flow A — Organization onboarding

A new user signs up, a profile and an organization are created, and the creating user becomes the administrator. Subsequent members join the existing organization. Partial onboarding is an explicit state the client handles.

Flow B — Connected inspection

The technician selects or registers equipment, optionally captures an image for identification, receives a suggestion, confirms or overrides it, answers the checklist, records notes and status, captures photo and location evidence, signs, and submits. Evidence objects upload to a private bucket under an organization-scoped path; the inspection row is inserted with references to them, and the database records the creation in the audit log.

Flow C — Offline inspection queue

When the network is unavailable the inspection is written to a local queue with an integrity digest and a visible pending state. On reconnect the queue replays, the digest is checked, evidence uploads, and the row is inserted. Successful items are removed only after confirmation. This is also an availability property: a server outage delays a field submission; it does not lose it.

Flow D — Corrective action

A failed inspection can raise a corrective action with an assignee and due date. Resolution evidence closes it.

Flow E — Data-subject erasure

A person's erasure request removes their personal references from the organization's records, including from the audit log, while the fact that an action occurred is preserved for the organization's accountability. Section 14 describes how that reconciliation works.

10. AI Assistance: Design, Providers, and Provenance

Provider independence

Three providers are supported behind one interface: Google, OpenAI, and Anthropic. Provider name, model identifier, API version, and request limits all come from a single typed configuration module, and an automated gate fails the build if any of them is written elsewhere. Changing provider or model is an environment change with no source edit. Commercially, that keeps pricing, availability, and regional posture a decision that can be revisited cheaply.

Trust boundary

Analysis is server-side; no client ever holds a provider credential. The handler authenticates before acting and fails closed when verification configuration is absent. A rate limit keyed on client address runs before verification to keep unauthenticated traffic away from the verification round trip; a second limit keyed on user identity runs after, and is the real meter. Both limits now remain correct when the service runs on many instances — see section 16.

Provenance and human authority

A suggestion is stored as a suggestion. The application preserves who supplied each fact, when, using which checklist, and whether the value was machine-suggested or technician-confirmed. The disclosure timestamp is generated by the server, because a client-asserted timestamp would look auditable while proving nothing.

AI-content marking in generated reports

The EU AI Act requires that AI-generated content be marked in a machine-readable way. Every generated inspection PDF now carries that marking in its document metadata, stating that AI assistance was involved and how. The marking was verified by rendering a real PDF and reading it back from the file bytes. That verification caught a defect no ordinary test would have: the PDF library silently ignored the metadata property its own type definitions declare, so every test that checked the object handed to the library passed while the actual file carried no mark. The fix writes the marking through both the declared and the library's actual path, and the test now reads the finished file. A compliance mark is verified in the bytes that leave the system, not in the object handed to a library.

11. Checklist and Content Configuration

Checklists are supplied by the content service and matched to an equipment class. Where no match exists the application falls back rather than blocking the inspection. Item state is deliberately narrow: pending, pass, or fail. There is no "not applicable" state anywhere in the code.

12. Photo, Location, and Signature Evidence

Photographs are downscaled on the client against a single declared size budget that both clients read from the same place. Location is captured where permitted. Signatures are captured on a drawing surface and exported as a fixed-size image with deliberate, non-themed colors, because that image is printed evidence rather than interface.

Evidence objects live in a private bucket with organization-scoped paths and storage policies keyed on the tenant segment of the path, so an object URL alone grants nothing.

13. Inspection Persistence and Data Model

Core records

Organizations, profiles, equipment, inspections, checklist answers, corrective actions, schedules, notifications, audit entries, consent records, plan limits, and data-subject requests.

Database-enforced behavior

- Every tenant-scoped table carries an organization identifier and a forced policy set.
- Signed inspection records are immutable — policies prevent update and delete rather than the client hiding buttons.

- Timestamps that matter for evidence are generated by the database.
- Role and tenant changes are refused by privilege-guard triggers.
- Plan limits are enforced in the schema, generated from a single declaration.
- Material changes are written to an append-only audit log by database triggers, described in section 14.

Eight versioned migrations define the schema from an empty project, and continuous integration proves on every change that the repository's migrations and the live project's applied migrations are the same set, in both directions.

14. Audit Log and Accountability

This is one of the most consequential pieces of the latest stage, because it sits exactly where two legal obligations collide.

The problem that was found

The audit table existed, but nothing wrote to it — so the system had no accountable change history at all. Worse, application users could insert rows into it, meaning any history it did contain could have been forged. And a latent defect sat underneath: the table was append-only, while GDPR erasure must remove a person's details from it. The moment anything began writing audit entries, erasure and account deletion would have crashed on the append-only guard. It had only stayed hidden because the table was empty.

What was built

- The database writes the log, not the application. Triggers on the organization and six core tables record every insert, update, and delete, with the acting user and — for changes that affect someone else, such as a role change — the affected person. Consent decisions are recorded with the specific decision made.
- No application role can write or edit it. Direct inserts by clients were removed. No application user, organization administrator, or support tool can alter an entry. Access is limited to reading an organization's own history.
- Erasure is permitted in exactly one shape. The append-only guard accepts a single kind of update: removing a person's identifying references — their user reference, network address, name, email, and affected-person reference — while every other field must stay identical, checked key by key. The record that an action happened survives; who it was does not. An administrator who changed the erased person's role keeps their own attribution on that entry.
- Deletion follows the organization. Entries are deleted only when the organization itself is deleted.
- Erasure is indexed on both the acting and the affected person, so it stays fast as the log grows.

Proof and honesty

Before the change, the audit test suite recorded eight failures and two passes. After it, 28 of 28 tests pass together with the tenant-isolation suite on a clean local database. The privacy policy was corrected in the same stage to avoid overclaiming: it states that no one using the service can edit an entry, and separately that the operators who maintain the underlying database hold technical access outside those application controls, restricted to authorised personnel. An earlier draft that said simply "no one can edit an entry" was caught by an independent review as too strong and replaced before it stayed live.

Status: the migration is complete and verified locally; applying it to the live database is a pending owner action, deliberately gated on two pre-checks against the hosted environment. As of 24 September 2026 it has still not been applied, and it cannot be until the hosted database is restored. *Update, 1 October 2026: applied to the hosted database after both pre-checks were re-run against it; the full suite, including the three erasure cases, passes 28 of 28 there. The same run also proved the organization-deletion case live: a test organization holding audit entries was deleted cleanly, which the append-only guard would previously have blocked.*

Lesson: *append-only and right-to-erasure are both absolute until they meet in the same database — decide the one mutation you permit before the first row exists.*

15. Module Ownership and Configuration Boundary

A single typed configuration module is the only place in the web client permitted to read environment variables, with no exceptions for server handlers or the deployment adapter. In the native client a single configuration module is the only place permitted to read compile-time definitions. An automated gate enforces this across 139 files, eight rules, zero violations.

Value class	Sole owner
Environment configuration, provider endpoints, model identifiers, API versions, limits, timeouts	One typed configuration module
Plan prices, periods, limits, feature descriptions	One plans module
Legal document content	Versioned content files
Actual secret values	The environment, plus a private version-control-excluded recovery file
Variable names and safe placeholder examples	One committed example file
Replica count, resource limits, health-check cadence, drain and connection timeouts	One deployment configuration file

The deployment generator also verifies that the server's built-in timeout defaults equal the values in the deployment configuration, so the two cannot silently disagree. Changing a provider, model, price, timeout, replica count, or legal text is a configuration or content edit, never a source edit.

16. Security Engineering

Security is layered deliberately, and each layer is named with what it can and cannot do. A condensed map of these controls is published with the repository so that anyone reporting a problem can say which control failed.

Tenant isolation and privilege

The database is the authorization boundary. Row-level security is enabled on all eleven tables and forced on the ten that hold tenant data — the eleventh holds shared, read-only plan limits — and is keyed on organization membership and role, resolved from the authenticated session. Storage policies are keyed on the tenant segment of the object path. Privilege-guard triggers refuse role and tenant changes. Cross-tenant reads, writes, and privilege boundaries are exercised by an authorization suite against the live database on every change, and by a local suite on a clean database built from migrations.

Evidence integrity and accountability

Signed inspections are immutable. Every material change is written to an audit log that no application role can forge or edit, as described in section 14. The database-written form of that log is verified locally and awaits application to the hosted database.

API protection

Handlers authenticate before acting, fail closed when verification configuration is absent, and never echo credentials. Rate limiting now works correctly across any number of server instances: counters live in a shared store, keys are hashed so no raw address or user identifier is written to it, a counter that loses its expiry is detected and repaired rather than locking a user out permanently, a remote store is refused unless reached over an encrypted connection, and any failure of the store falls back to per-instance limiting — never to no limiting. Fallbacks are logged at a throttled rate so an outage does not flood the logs. Each of these behaviors has a unit test, including two replicas sharing one budget.

Server hardening

Request header and body timeouts are now genuinely enforced. A verification test revealed that the Node runtime checks those timeouts only on a background interval that defaults to thirty seconds — so a configured fifteen-second header timeout was really fifteen to forty-five seconds, leaving room for slow-client resource exhaustion. The interval is now derived from the configured timeouts, and a slow client is cut off with a timeout response on schedule. Request bodies are bounded. In the orchestrated form the container runs non-root with a read-only filesystem and all capabilities dropped, and a network policy admits traffic

only from the ingress.

Transport and browser isolation

Strict transport security, content security policy, frame denial, and cross-origin opener and resource policies are declared once and generated for every deployment target. The production check verifies that all nine headers served live are identical to that single declaration.

Authentication posture as code

Authentication provider settings — password rules, session behavior, email and rate limits — are captured as a versioned baseline and checked for drift, so a setting changed in a web console is detected rather than silently trusted. Known gaps are listed in that baseline so they are recorded rather than rediscovered.

Vulnerability disclosure

A published security policy invites private reporting, sets good-faith response targets, grants safe harbour to researchers who follow the rules, and defines scope. An industry-standard security contact file is served from the live site so automated tooling and researchers can find the reporting channel. The repository's private vulnerability-reporting channel, which the policy names, was found switched off in September and has been enabled.

Supply chain and static analysis

Three independent scanning layers run over the repository: a scanner on every file edit, a pre-commit stage running secret detection, Python static analysis, and a security-audit ruleset, and a continuous-integration workflow that repeats secret detection and the security-audit rules. None may be bypassed. Dependency updates arrive as automated update requests for packages and build actions, and each must pass the full pipeline before it is merged. An earlier edition of this document described a separate dependency-review job; there is none, and the statement has been corrected. The container base image is pinned by digest.

Published threat model

The public threat model lists the assets worth protecting, the trust boundaries between the browser, the mobile app, the server handlers, the database, storage, and external providers, and a STRIDE table giving each threat its control and residual risk. It names its known gaps instead of implying there are none — for example, that sign-in has no CAPTCHA yet, so credential stuffing is only rate limited, and that the shared authentication email sender caps password resets at a few messages per hour.

What is not claimed

No independent security assessment, penetration test, or control audit has been performed. Everything above is engineering and automated verification, not third-party attestation.

17. Legal and Regulatory Posture — US and EU

This section states positions and the engineering behind them. It is not legal advice, it is not a certification, and the legal documents have not yet been reviewed by counsel.

Terms of Service

The Terms of Service were rewritten as a versioned content document with a fixed effective date — replacing a page whose "last updated" date was generated from the visitor's clock and that cited the wrong rules. Sixteen sections cover acceptance, the service, accounts and organizations, AI-assisted analysis, electronic signatures, the customer's own regulatory obligations, acceptable use, plans and billing, data, availability, warranties, limitation of liability, suspension and termination, governing law and disputes, changes, and contact. Two positions are worth highlighting:

- Electronic signatures are framed against the US federal ESIGN Act, the Uniform Electronic Transactions Act adopted by US states, and the EU eIDAS Regulation, with the customer's responsibility for any transaction requiring a higher signature standard stated plainly.
- Workplace-safety inspections are framed against the specific US occupational safety rule for monthly visual inspection of portable fire extinguishers. The Terms are explicit that the product records inspections and does not replace the employer's legal duty, and that no uptime commitment is made.

Legal-entity details are marked as placeholders to be completed by the business before commercial launch.

Inspection records against the safety rules

The US occupational safety rule requires portable extinguishers to be visually inspected monthly and annual maintenance to be recorded and retained. The product schedules recurring inspections per equipment item and flags overdue ones; every record carries the inspector, the date, and the findings, with timestamps generated by the database rather than the device. The NFPA 10 standard, in the edition adopted by the local authority, requires monthly inspection records to be kept for at least twelve months; that floor is encoded in the schema, and signed records cannot be edited or deleted. The product records evidence of inspections — it does not perform them or certify equipment, and the employer's own duty is unchanged.

EU AI Act

A written classification position records why the AI use is assessed as it is and what would change that assessment. Two transparency duties are implemented rather than planned: people are told when they are interacting with AI assistance, in both clients, with a server-generated disclosure timestamp; and AI-involved content is marked in a machine-readable form inside every generated PDF report, as described in section 10. These transparency duties under Article 50 apply from 2 August 2026.

GDPR and data residency

The database was provisioned in an EU region deliberately, so EU personal data does not leave the EU. A record of processing, a data-processing position, a sub-processor list, and a breach-response procedure accompany the AI Act position; the record of processing was updated to describe the audit log accurately, including operator access. In September two public services without a processing agreement were added as separately listed recipients: reverse geocoding, which receives inspection coordinates and the requester's network address, and a breached-password check, which receives the network address and a five-character prefix of a password hash — never the password. The privacy policy's recipient summary has yet to name them.

Accountability and the right to erasure

The database-written audit log provides accountability for material changes, and erasure is designed into it rather than bolted on, as section 14 describes. Account deletion and organization deletion were verified not to be blocked by the log.

Consent

Optional tags are structurally incapable of loading before consent is given; the gate is a property of how the page is assembled, not a script racing to run early. Consent is not requested on pages that store nothing optional, and even a small convenience — remembering that the intro animation was already shown — is not stored without preference consent.

Consent to sign electronically is a separate matter from storage consent. The web client requires an explicit "I agree to sign electronically" action before the signature pad accepts input; the mobile client currently shows a notice, and an explicit control is on the roadmap. *Update, 1 October 2026: the mobile client now has the same explicit control, and consent cannot be withdrawn while a signed record is being filed. Neither client yet stores the consent itself as a separate piece of evidence; that is on the roadmap.* On both, the signature is bound to the signed-in account and the inspection record, and signed records are immutable. The US electronic-signature rules look for intent to sign, consent to do business electronically, attribution, and an accurately reproducible record; the EU rule for simple electronic signatures is that one may not be refused legal effect solely because it is electronic. Qualified electronic signatures are not offered.

Data-subject requests

A public intake page and a dedicated handler exist, backed by a migration, so a request is a record with a lifecycle rather than an email. The intake page, previously missing from the live site, is now deployed.

International transfers

Primary data stays in the EU. Transfers to US AI providers rely on standard contractual clauses and, where the provider is certified, the EU–US Data Privacy Framework. The framework is valid law — the EU General Court upheld it on 3 September 2025 — but an appeal is pending before the Court of Justice, which

invalidated both of its predecessors, so standard contractual clauses are kept as the fallback basis rather than relying on the framework alone. The EU's adequacy decision for the United Kingdom has been renewed to 2031.

US state privacy and Global Privacy Control

Roughly twenty US states have comprehensive privacy laws in effect in 2026. The product offers rights-request intake, does not sell or share personal data, serves no targeted advertising, and runs no analytics or tracking scripts at all. The Global Privacy Control browser signal is detected and honoured as an opt-out, as required in California, Colorado, Connecticut, Delaware, Maryland, Minnesota, Montana, Nebraska, New Hampshire, New Jersey, Oregon, and Texas; because California requires a visible confirmation from 1 January 2026, an "Opt-Out Preference Signal Honored" notice is shown. As a business-to-business processor, most state obligations reach the product through customer contracts, and service-provider contract templates are on the roadmap.

EU Cyber Resilience Act

The Flutter mobile application is a product with digital elements once it is made available on the EU market, and its backend is likely in scope as the app's remote data processing; pure software-as-a-service is generally excluded. Vulnerability and incident reporting duties apply from 11 September 2026, including to products already on the market, and the full essential requirements from 11 December 2027. Reporting runs through the EU single reporting platform: an early warning within 24 hours, a notification within 72 hours, and a final report 14 days after a fix is available for an exploited vulnerability or one month after notification for a severe incident. A coordinated disclosure policy and dependency scanning exist today; a software bill of materials per release, a reporting runbook, and conformity documentation are planned before any store release. The app is not yet published.

NIS2 Directive

Cloud and software-as-a-service providers fall within the directive's sectors, but only from medium size under the EU definition — fifty or more staff, or annual turnover and balance sheet both above ten million euros. The product is below that threshold today, so the directive does not apply directly; customers that are in scope may pass supply-chain security terms down by contract.

Accessibility law

Contrast, target size, landmark structure, heading order, keyboard focus, and reduced-motion behavior are checked, and specific failures were found and fixed. Full conformity across every screen is not claimed.

The legal picture behind that work: the US Americans with Disabilities Act has no federal web rule for private businesses, and courts are split on when a website alone is covered, but where it is, WCAG 2.1 AA is the usual benchmark. For state and local government — customers such as municipal fire departments — web content and apps must meet WCAG 2.1 AA by 26 April 2027 or 26 April 2028, depending on size, and vendors are asked to prove it at procurement. The European Accessibility Act does not reach purely business-to-business services, and microenterprises are exempt from its service requirements; WCAG 2.1 AA and the matching European standard remain the target regardless. The public accessibility statement describes the product as partially conformant, self-assessed, with its known gaps listed; a third-party audit is on the roadmap.

Documentation honesty as a legal control

Claims that a regulator or client could hold against the business were removed from public materials during this stage: an unsupported five-year retention claim, an unearned "SOC 2 ready" label, a reference to a mobile technology no longer used, and an unsourced statistic. Overclaiming in public documentation is treated as a compliance defect, not a copywriting choice. The same discipline was applied again in late September: before the new public documents were published, an independent review found overclaims — several inherited from older documents — and each was corrected, including in this document.

18. Subscription and Billing Behavior

A webhook handler receives subscription events. The payment integration is currently unconfigured, and the handler says so explicitly rather than failing in a way that looks like a transient error. Placeholder credentials are never shipped. Customer lookup, database-result validation, plan mapping, and event idempotency must be verified before any commercial launch.

Update, 1 October 2026: a defect was found and fixed before the integration was ever configured. On the serverless hosting target the platform parses the request body before the handler runs, so the exact bytes the payment provider signed were lost and every genuine event would have been rejected. The handler now verifies the exact original bytes on both deployment targets, refuses rather than re-serialises a body that was already parsed, and logs only the error message — the provider's error object carries the whole event, customer details included. A credential still holding the example file's placeholder shape is treated as unconfigured. Signature handling is tested with provider-signed payloads; subscription reconciliation is still untested.

19. Reports, Notifications, Equipment, and Scheduling

Inspection records render to PDF in the browser with embedded provenance marking. Notifications are delivered through realtime events. Equipment and schedule views are organization-scoped. Team and Reports remain placeholders.

20. Failure Behavior

- Missing session or profile resolves to an explicit partial state, not a blank screen.
- Network loss during an inspection routes to the local queue with a visible pending indicator.
- Content service failure falls back rather than blocking the inspection.
- AI provider failure returns a clear error; the inspection proceeds without a suggestion.
- An unconfigured integration answers honestly instead of simulating an outage.
- A failed shared rate-limit store degrades to per-instance limits, never to unlimited access.
- A slow or stalled client is cut off by enforced timeouts instead of holding a connection indefinitely.
- A replica being replaced reports itself not-ready and drains before closing.
- The 3-D hero falls back to a complete still-and-list composition when 3-D rendering is unavailable, and never shows unpositioned markers while loading.
- Forms and primary actions render visible from the markup and are never gated behind an animation.

21. Scalability: From One Container Toward One Million Concurrent Users

This section is written to be checked. It separates what is measured, what is built, and what the architecture is designed for, and never mixes them. The current deployment serves none of the large numbers below and claims none of them. What the codebase shows is that getting from here to there is configuration and provisioning, not a rewrite.

Why the architecture scales — three properties

1. Static export. Pages and the application shell are cacheable bytes served from the edge. With content-hashed assets and long cache lifetimes, most traffic never reaches an origin.
2. Stateless origin. No session state lives on the server, so any number of replicas run behind a load balancer with no sticky sessions. The one piece of shared state an origin needs — rate-limit counters — now has a shared store.
3. Authorization in the database. Row-level security is enforced on every query whichever replica issued it. Adding replicas, regions, or read replicas cannot weaken tenant isolation. This is what makes horizontal scaling *safe*, not merely possible.

Request path at scale

Browsers and the mobile app reach a content delivery network edge that serves static pages, scripts, styles, and media without touching the origin. Cache misses and API calls go to a health-checked load balancer — the reverse proxy's least-connections pool or a Kubernetes service — which spreads them across one to fifty stateless Node origin instances that share one rate-limit store. Those instances reach the EU database through its connection pooler, with a read replica for lag-tolerant reads. The database is the binding constraint, and the plan is built around that fact.

Measured

Single process, one machine, static serving path:

Concurrency	Throughput	p50	p95	p99	Failed
25	3,472 req/s	6.7 ms	11.5 ms	14.5 ms	0
100	3,619 req/s	26.8 ms	39.4 ms	48.2 ms	0
300	3,657 req/s	79.4 ms	106.9 ms	189.0 ms	0

Flat throughput with latency rising in proportion is the signature of a saturated, CPU-bound single process, and it checks arithmetically: 3,657 requests per second multiplied by 79.4 milliseconds is about 290 requests in flight, matching the 300 offered. This is not an application capacity — no database or AI call is in this path. It establishes that the origin's bottleneck is one CPU core, the cheapest thing in the system to multiply.

Capacity model per tier

A concurrent user is a person with the app open, not a request. The model assumes one page or asset request every two seconds while active, at least ninety percent of those served from the edge cache once warm, one percent of users mid-database-query at any instant, and one in ten with a live dashboard socket. Those assumptions are stated so they can be argued with.

	10,000 concurrent	100,000 concurrent	1,000,000 concurrent
Offered HTTP load	~5,000 req/s	~50,000 req/s	~500,000 req/s
Reaching the origin at 90% edge hit	≤ 500 req/s	≤ 5,000 req/s	≤ 50,000 req/s
Origin instances, with 50% headroom	1 needed → run 3 for redundancy	~3 → run 3–6	~28 → autoscale toward 50
Concurrent in-flight database queries	~100	~1,000	~10,000
Database tier	Pro-class pooler, read replica for aggregates	Larger compute and replicas, connection budget reserved for writes	Beyond one database primary — tenant sharding
Realtime sockets	~1,000	~10,000	~100,000
Origin side already built	Yes	Yes	Yes, multi-node

Origin instance counts are extrapolated from the measured single-core figure. Database rows are designed-for estimates.

The order things break in

1. Realtime connections — two hundred on the free database plan.
2. Pooled database connections — two hundred on the free plan; concurrent inspection submissions reach it first.
3. Authentication email — the shared sender allows very few messages per hour, so password resets stop long before load does.
4. AI provider quota — limited per account at the provider, which is why analysis is rate-limited per address and per account.
5. Reverse geocoding — the public geocoding service's usage policy allows about one request per second and no heavy use. Its address is configuration; at scale it must point at a self-hosted or contracted geocoder. *(Added in late September.)*
6. Origin CPU — last, and cheapest to fix.

The ladder — what each step lifts and what is already built

Step	Lifts	Already built	Change required	Cost
0. Current	—	Static export, edge caching, row-level security, rate limits, offline queue	—	None
1. Multiply the origin	Origin CPU	Replica count in one config file generates N containers in a least-connections pool with readiness checks; drain on deploy	Set the replica count and regenerate	None on an existing host
2. Shared rate limiting	Limits multiplied by replica count	Shared store adapter with hashed keys, expiry repair, safe fallback; unit tested with two replicas	Point one setting at a store	Free tiers exist
3. Database plan	Pooler, realtime, storage, backups	Tier-aware configuration	Environment variables	Paid plan
4. Read replica	Read load on the primary	Read client that keeps row-level security, dashboard aggregates routed to it	One environment variable	Paid replica
5. Orchestration	Single host, manual failover	Generated Kubernetes deployment: zero-unavailable rolling updates, startup/readiness/liveness probes, non-root read-only containers, autoscaling 3→50, disruption budget, network policy — schema-validated and drift-checked in CI	A cluster	Paid cluster
6. Capacity test	"Designed for" becomes "measured"	Load-test plan with smoke, 10k, 100k, and 1M profiles and pass/fail SLO thresholds; refuses a remote target without explicit opt-in	A provisioned environment and load generators	Generator compute

What is genuinely hard beyond that

At sustained seven-figure concurrency the problem stops being configuration. A single database primary becomes the write bottleneck, and the plan is to shard tenants across database projects — the schema is already keyed on organization everywhere, which is the prerequisite — move evidence photos to dedicated object storage with its own delivery network, and replace per-dashboard realtime sockets with a purpose-built fan-out service. None of that is built. The codebase is shaped so it can be done without changing the authorization model.

What has not been measured, stated plainly: the database tier and any multi-replica throughput have not been load tested. Running such a test against a free-tier project would be a denial of service against ourselves; it needs a provisioned environment. Anyone presenting a finished one-million-user plan without having measured ten thousand is guessing — which is why the first rung here is real and the rest are labelled as design.

The roadmap — phases with exit criteria

The published roadmap turns the ladder above into phases. Each ends with an exit criterion that has to be measured, not a date; anything that costs money needs the owner's approval before it starts.

Phase	Scope	Exit criterion
0. Built	Everything this document describes as built or verified	—
1. Production readiness	Restore the hosted database; apply the audit-log migration and authentication baseline; dedicated authentication email; CAPTCHA on sign-in and sign-up; counsel review and the operator's legal identity; name the two third-party services in the privacy policy; explicit signing consent on mobile; a web test suite; an automated check that every security-policy origin is a listed recipient; payments in test mode; one restore drill	Every item closed, and the production check green for 30 days
2. Mobile stores and the Cyber Resilience Act	Google Play release with an off-site keystore backup; iOS signing and App Store release (paid); a software bill of materials per release; the reporting runbook; an accessibility audit and public conformance statement	Both apps in their stores, a bill of materials per release, the runbook rehearsed once
3. Tier 1 — 10,000 concurrent, 99.9% target	Three origin replicas; the shared rate-limit store; a paid database plan; a read replica; a self-hosted or contracted geocoder; a one-minute external uptime monitor; the 10k load profile against staging	The 10k profile passes its thresholds, and 30 days at 99.9% or better on the external monitor
4. Tier 2 — 100,000 concurrent, 99.95% target	Multi-node Kubernetes; larger database compute and more replicas; point-in-time recovery; a third-party penetration test; SOC 2 Type I readiness; the 100k profile from distributed generators	The 100k profile passes, a restore drill meets the recovery objectives, and penetration-test findings are closed
5. One million and beyond	Tenant sharding; evidence on dedicated object storage and its own delivery network; purpose-built realtime fan-out; multi-region with EU data staying in the EU; SOC 2 Type II, single sign-on, customer-managed retention	The 1M profile passes with its environment named, and multi-region failover is rehearsed

Until a phase's criterion is measured, its figures stay design targets and are described as such.

22. Availability Engineering and Service-Level Objectives

Objectives per tier — design targets

Tier	Deployment	Availability objective (30 days)	Monthly error budget	Latency objective
0. Current	One container, shared host, free database	None claimed	—	—
1. Redundant origin	Two or more replicas behind the health-checked pool, paid database plan	99.9%	43 min 12 s	p95 under 500 ms for pages, under 300 ms for API rejections
2. Orchestrated	Kubernetes, 3–50 pods across nodes, disruption budget, paid database plus read replica	99.95%	21 min 36 s	Same

For reference, 99% availability allows 7 hours 12 minutes of downtime a month. That figure sits comfortably inside the tier-1 target — but it remains a *target* until a tier-1 deployment has run for a month and its monitor confirms it. The ceiling no application can engineer past is the managed database provider's own availability; any end-to-end commitment must be checked against the provider's published agreement for the plan in use.

Mechanisms that keep requests from failing — verified

- Liveness versus readiness. A dependency-free liveness probe answers "can this process serve at all", so a database blip never restarts every replica at once. A separate readiness probe answers "should traffic be sent here" and turns unavailable during shutdown, because a draining replica is healthy and must not be restarted.
- Graceful drain. On a stop signal the instance reports not-ready while still serving, long enough for the load balancer's health check to notice; responses carry a connection-close instruction so keep-alive clients move elsewhere; then the listener closes, idle connections close immediately, in-flight requests finish, and a hard deadline ensures one hung request cannot block a deploy. Verified 13 of 13 in the production container image.
- Timing invariants. The proxy's keep-alive is shorter than the server's, so the proxy never reuses a socket at the instant the server closes it — the cause of intermittent errors no log explains. The orchestrator's stop grace exceeds drain plus deadline, so an instance is never killed mid-drain. The generator refuses any configuration that breaks either ordering.
- Honest scope. Zero-downtime deployment is a property of tier 1 and above, where another replica exists to take traffic. On the current single container, a deploy still drops requests for a few seconds; this is stated rather than hidden.

Redundancy by layer

Layer	Tier 1	Tier 2
Origin	N replicas, active health checks, retry within a bounded window	Rolling updates with zero unavailable, spread across nodes, disruption budget of two, autoscaling 3→50
Rate limiting	One shared budget across replicas, safe fallback	Same
Static assets	Edge cache keeps serving through an origin outage	Same
Database	Managed, daily backups on paid plans	Plus read replica
Field technician	Inspections queue offline and sync on reconnect	Same

Failure modes and recovery

A written failure-mode analysis covers a crashed replica, rolling and single-replica deploys, a total origin outage, database outage and connection exhaustion, rate-limit store outage, AI provider outage, slow-client attacks, free-tier database pausing, and a wrong backend compiled into the site — each with detection, automatic response, user impact, and a numbered runbook. Recovery objectives are stated per tier: on the paid tier, data loss bounded by daily backups and origin restore within an hour; on the orchestrated tier with point-in-time recovery, data loss of minutes and origin restore within fifteen minutes. No restore drill has been run yet, so recovery times remain targets.

Production monitoring

A automated production check runs against the live deployment and asserts that the site answers, that liveness and readiness pass, that all nine security headers match their single source, that the API is mounted and rejects the wrong method, that the compiled site points at the intended backend, and that the backend exists. It is designed to run every thirty minutes in continuous integration. It exists because of a real incident: the live site once returned success on every page for about a day while its backend had been removed and nobody could sign in. A successful page load is not a health check. The check passed 7 of 7 against the current deployment; enabling its scheduled run needs one repository secret, recorded as a pending owner action. *Update, 24 September 2026:* the scheduled run was switched on with the late-September publication. Its first scheduled run had not yet taken place at the time of writing; while the hosted database is unreachable it will report failure, which is the purpose it was built for. *Update, 1 October 2026:* it reported the outage as designed, was paused at the owner's request while the database was down to stop repeated failure notices, and was re-enabled when the database returned; it passes 7 of 7. Its run history shows the hosting service starts the thirty-minute schedule every two to five hours in practice, so detection time is measured in hours.

23. Test and Verification Evidence

Checks executed automatically on every change

- Static gates — configuration-boundary enforcement across 139 files, plan-limit generation check, Kubernetes manifest drift check, unit tests, lint, TypeScript checking, static export, and server bundle. Lint completes with zero errors and six warnings against a threshold of twenty. As of late September it reports eight: a lint rule introduced by a framework dependency update flags two existing navigation calls. *As of 1 October: configuration-boundary enforcement across 157 files and nine rules, now including the check that every external origin in the content security policy is named in the sub-processor list; 104 unit tests.*
- Native client, Android — formatting, strictest static analysis, 423 tests, and a release build, with release hardening asserted against the packaged manifest and fail-closed signing verified by confirming an unsigned artifact does not verify. *429 tests as of 1 October.*
- Native client, iOS — release compile on macOS hardware.
- Live service gates — dependency reachability, generated-type drift, tenant isolation and privilege boundaries against the live database, migration agreement in both directions, and the database vendor's security and performance advisors. Failing since 24 September because the hosted database is unreachable. *Passing again from 1 October, including the audit-log suite against the hosted database.*
- Security scanning — secret detection, static analysis, and security-audit rules. Dependency updates arrive as automated update requests that must pass every job; an earlier edition listed a dependency-review job, which does not exist.

Checks executed in this stage

Check	Result
Unit suite: consent, rate limiting, shared-store failure and repair, report provenance	43 of 43
Database suite on a clean local database: tenant isolation plus audit log, including erasure	28 of 28 (previously 8 failing, 2 passing)
Server lifecycle in the production container image	13 of 13
Real rendered PDF carries AI-content marking	Pass
Deployment generator invariants and Kubernetes drift	Pass
Load-test plan script validation	Valid
Post-deploy checksum parity, built versus served	143 of 143 identical
Automated production check against live	7 of 7
Live browser check of the hero loading sequence	Markers hidden until ready, then placed on the model
Independent fresh-context code review of the stage	Ready with fixes, zero blockers; all fixes applied and re-gated

Checks executed in the late-September stage

Check	Result
Web unit suite on Windows and on a clean Linux container matching the build runner	43 of 43 on both (0 of 43 on Linux before the fix)
Configuration boundary	139 files, zero violations
Lint	Zero errors, eight warnings, threshold twenty
Static export and server bundle	Pass
Production dependency audit	Zero known vulnerabilities
Native client test suite	423 passing
Continuous integration on the latest commit: web static gates, Android, iOS	Pass
Continuous-integration security scanning	Pass
Continuous-integration live-services job	Fail — hosted database unreachable
Secret scan of every published commit	Clean
Relative links across the published documentation	Zero broken
Independent review of the new documentation before publication	*Not ready*, six major findings; all fixed before publication

Checks executed in the early-October stage

Check	Result
Web unit suite on Windows and on a clean Linux container matching the build runner	104 of 104 on both
Payment webhook with provider-signed payloads over a real HTTP server	Pass on every delivery shape
Self-hosted server, built bundle: signed, tampered, and re-serialised webhook	Accepted, rejected, rejected
Tests proven non-vacuous: run against the previous webhook handler, and with the mobile consent gate forced off	Previous handler fails 3 of 12 webhook tests; gate off fails 2 consent tests
Disclosure check: planted policy violations	7 of 7 caught
Native client: formatting, static analysis, tests	Clean, no issues, 429 passing
Audit-log migration pre-checks against the hosted database	Both pass; applied

Check	Result
Tenant isolation and audit suite against the hosted database	28 of 28
Migrations applied and tracked	8 of 8
Erasure function callable only by the server role	Verified by privilege query and by a real anonymous call, which is refused
Continuous integration, all four jobs, latest commit	Pass
Live deployment checksum parity, built versus served	144 of 144 identical
Automated production check	7 of 7
Independent fresh-context review	Ready with fixes, zero blockers; fixes applied and re-gated

Checks not executed

- No unit or browser end-to-end suite for the web client's user journeys.
- No database-tier or multi-replica load test.
- No month-long availability measurement on a redundant tier, and no restore drill.
- No physical-device camera, location, offline, background, or installation test.
- No payment webhook replay or subscription reconciliation test. *1 October: signature handling is now tested with provider-signed payloads; subscription reconciliation is still untested.*
- No AI provider contract test against a live provider.
- No independent security assessment, penetration test, control audit, or counsel review of the legal documents.
- No automated check that every origin in the content security policy appears in the sub-processor list.

24. Test Interpretation — Lessons That Became Practice

The strength of this evidence comes from where the checks run and from whether they can fail. Several lessons from this project are now standing practice:

- A gate that has never failed has never been tested. One live gate was found incapable of reporting a failure at all. New gates are now given bad input once and confirmed to fail before they are trusted.
- A test is not evidence until it has failed for the right reason. The server-lifecycle test was run against a deliberately unfixed build and failed on exactly the one check the fix addressed, 12 of 13 versus 13 of 13. Earlier, a test bug and a clock jump inside a virtual machine had briefly made correct code look broken.
- Verifying a check yourself is not running it where it matters. Tooling produced different output formats depending on what invoked it; checks now pin their expected input format.
- Green continuous integration measures the repository, not the product. Every gate passed while the live site could not sign anyone in, because none of them made a single request to production. That is why the automated production check exists.
- A lesson without a gate is a lesson learned twice. The wrong-backend outage happened twice, thirteen days apart, before a check was built for it.
- Setting a timeout and enforcing a timeout are two different settings. Section 16 explains the runtime detail.
- A suite that has only run on one operating system has been verified on that operating system. The web unit suite passed 43 of 43 on the development machine and 0 of 43 on the build runner, because a path that is a script on Windows is a native binary on Linux. A new build step is now run in a container matching the runner before it is pushed.
- A document that says it can be checked against the code needs a check that does it. The sub-processor list claimed to match the content security policy; the first time anyone compared them, two services were missing.
- Public documentation is code. Every claim in the new documents was checked against the source before publication, and the independent review still found overclaims inherited from older documents.

25. Operations and Release Model

Web release

Install locked dependencies, supply the target deployment's public configuration, run the gates, build the static client, and publish. Public configuration values are compiled into the static output, so a build must be given its deployment's values — a default build ships development values to production, and a backend change is a mandatory redeploy, not an environment tweak.

Self-hosted release

The release procedure, as executed for the current deployment:

1. Drift check first. The files currently served are hashed and compared against the record of the previous release, proving nobody changed production directly.
2. Build locally with production public configuration, bundle the server, and generate the proxy site, composition, and orchestration manifests for a new release identifier.
3. Upload to a fresh release directory as an unprivileged account, never overwriting a previous release.
4. Write runtime secrets from current values into a root-only file — never copied from an older release, because an old file can carry a retired backend into a correct build.
5. Build and start the container, waiting until it reports healthy.
6. Swap the proxy site with a backup taken first; the proxy configuration is validated before reload and the previous file is restored automatically if validation fails, protecting the other sites on the shared host.
7. Prove parity — every file checksum-compared between the built release and the served one — and run the automated production check.

Previous releases are retained for rollback, and the rollback steps are recorded with each deployment.

Database and storage

Migrations apply in versioned order from an empty project; continuous integration proves applied and tracked migrations agree in both directions; the vendor's advisors run as gates. Schema changes to the live database are applied deliberately and separately from code deploys, after pre-checks against the hosted environment.

Orchestrated release

The generated Kubernetes manifests apply as one unit: namespace, deployment, service, autoscaler, disruption budget, and network policy. Rolling updates keep zero instances unavailable, and the readiness probe gates traffic to each new instance.

Mobile release

Both platforms compile in continuous integration. Android release signing fails closed and is verified by signature, not filename. iOS is compiled unsigned only. Neither platform has been published. Before a store release the roadmap adds a software bill of materials per release and a Cyber Resilience Act reporting runbook, because the reporting duties apply as soon as the app is on the EU market.

Observability

Monitor availability through the automated check, readiness transitions during deploys, authentication failures, policy denials, rate-limit fallbacks, storage errors, queue backlog, content fallbacks, AI latency and provider failures, webhook retries, and realtime reconnects. Logs carry support-safe identifiers and omit credentials and evidence content.

26. Current Limitations and Risk Register

Closed across the September editions, each verified by an automated check:

- ~~Critical authorization: self-service role escalation.~~ Refused by a database trigger.
- ~~High evidence privacy: public storage bucket.~~ Private bucket with organization-scoped policies.
- ~~High authentication: analysis handler fails open.~~ Fails closed.
- ~~High accountability: audit log empty and forgeable; erasure would crash once written.~~ Database-written, unforgeable, erasure-compatible; verified locally, live application pending.
- ~~High operations: live site served a removed backend with no alert.~~ Rebuilt on the current backend; automated production check detects it.
- ~~Medium rate limiting multiplied by replica count.~~ Shared store with safe fallback.
- ~~Medium slow-client timeouts not enforced on schedule.~~ Enforced; verified in the production image.
- ~~Medium public documentation overclaims.~~ Corrected.
- ~~Medium hero loading sequence showed unpositioned markers.~~ Hidden until ready; verified live.
- ~~Medium deployment reproducibility.~~ Eight migrations build the schema from nothing.
- ~~Medium disclosure channel: the security policy pointed to a private reporting channel that was switched off.~~ Enabled in September.
- ~~Medium test portability: the web unit suite passed on the development machine and failed on the build runner.~~ Fixed; runs on every change.
- ~~Medium documentation overclaims inherited by earlier editions: forced row-level security on every table, a dependency-review job.~~ Corrected in place.

Open, stated without softening:

- High service availability. As of 24 September 2026 the hosted database the live build points to is unreachable, so sign-in on the live site fails until it is restored and the site rebuilt against it. *Resolved 1 October 2026: database reachable, site rebuilt and redeployed, 7 of 7 production checks.*
- Medium privacy disclosure. Two public services without processing agreements are listed in the sub-processor list but not yet named in the privacy policy, and no automated check yet compares the content security policy with that list. *1 October: the automated comparison now exists and runs in every build; the privacy-policy text still has to name both services.*
- Medium signing consent on mobile. The mobile client shows a notice rather than an explicit consent-to-sign control. *Resolved 1 October 2026: explicit control added.*
- Medium consent evidence. Both clients ask for consent to sign electronically, but neither stores it as a separate record, so it cannot be produced independently later (found 1 October).
- Medium test isolation. The live test suite runs against the production database project and, by design, leaves one test organization per run, because it signs records that can never be deleted. A separate test environment is the fix (found 1 October).
- Medium held dependency updates. Two major-version updates — the icon library and one AI provider's software kit — are held until tests can catch their failure modes.
- High billing integrity. Payment integration unconfigured; must be verified before commercial launch.
- High legal review. Terms and privacy policy are engineered against the named frameworks but have not been reviewed by counsel, and legal-entity details are placeholders.
- Medium capacity evidence. The database tier and multi-replica throughput are not load tested; large-scale figures are design targets.
- Medium availability evidence. No redundant tier has run for a measurement period and no restore drill has been performed; the current single container claims no uptime figure.
- Medium pending live changes. The audit-log migration and the authentication-settings baseline await application to the live environment; the scheduled production check awaits one repository secret. In late September the scheduled check was switched on; the migration and baseline still wait, now behind the database restoration. *1 October: the audit-log migration is applied; the authentication-settings baseline still waits for a write-capable credential.*
- Medium web test maturity. No unit or browser end-to-end suite for the web client's user journeys. *1 October: 104 unit tests now cover the web libraries and server handlers; there is still no browser user-journey suite.*
- Medium offline durability. Retry semantics and signature preservation across a failed replay remain incompletely verified.
- Medium checklist evidence. Fallback content may not match the equipment class, and the checklist version is not preserved with the record.
- Medium device verification. No physical-device testing on either platform.
- Low loading polish. The hero's placeholder image frames the model differently from the loaded 3-D scene, and the intro screen appears a moment after first paint; changing either is a design decision awaiting the owner.
- Product completeness. Team and Reports are placeholders.

27. Developer Handoff

Source-of-truth order

Use the migration history for database intent, the server handlers for protected integration behavior, the deployment configuration for runtime and scaling behavior, the application modules for user flows, the continuous-integration definition for what must pass, and this document as a reconciled map. When they disagree, verify the deployed contract and correct both code and documentation. Documentation that disagrees with the code is worse than absent documentation, because it is trusted.

Safe change discipline

- Preserve tenant identity from authenticated server or policy context; never trust a client-supplied role or organization.
- Version data-contract changes and test migration order from a clean project.
- Decide the permitted mutation shape before adding any append-only record.
- Keep the origin stateless; anything shared between replicas goes in a shared store with a safe fallback.
- Change replica counts, timeouts, and probes only in the deployment configuration and regenerate; never hand-edit generated proxy or orchestration files.
- Keep liveness free of dependencies; put dependency checks in readiness.
- Verify compliance marks and security headers in the bytes that actually leave the system.
- Give anything positioned by a render loop an explicit not-yet-positioned state.
- Never gate a form or primary action behind an entrance animation.
- Prove a new gate can fail before trusting it, and pin the format of any tool output it parses.
- Keep public documentation free of credentials, internal locations, personal data, environment-specific endpoints, and unearned claims.

Minimum release evidence

Continuous-integration results for the changed area, migration agreement, authorization and audit tests, lifecycle tests where server behavior changes, checksum parity after deploy, and a passing automated production check — plus physical-device results where the change touches device behavior.

28. Representative Operational Traces

Successful connected inspection

session -> profile and tenant -> equipment -> checklist camera -> optional suggestion -> technician confirmation answers -> location -> signature -> private evidence upload inspection insert -> audit entry -> dashboard and marked PDF report

Recoverable offline inspection

network failure -> local queue -> visible pending state reconnect -> digest check -> upload -> database insert confirmed success -> remove local item

Zero-downtime replica replacement (tier 1 and above)

stop signal -> readiness reports not-ready -> load balancer stops routing in-flight requests finish -> listener closes -> clean exit new replica starts -> readiness passes -> traffic resumes

Data-subject erasure

verified request -> personal references removed from profile audit entries stripped of identifying fields only -> action history preserved

29. Client Review Checklist

- Confirm which equipment classes and inspection standards the product is authorized to represent.
- Approve the exact checklist for each equipment class and the behavior when no match exists.
- Define whether photo, signature, precise location, device metadata, and AI processing are required, optional, or prohibited.
- Approve user roles and who may close corrective actions.
- Define retention and deletion rules for inspections, PDFs, photographs, signatures, and audit history.
- Complete the legal-entity placeholders and have counsel review the Terms of Service and privacy policy for each market served.
- Confirm the wording that explains AI assistance and technician responsibility.
- Choose a target concurrency and availability commitment so the scaling ladder and service tiers in sections 21 and 22 can be provisioned and measured against a real number.
- Decide the database plan tier, which unlocks backups, higher connection limits, and the first availability target.
- Decide whether store distribution of the native applications is in scope.
- Review free-plan limits and paid entitlements, noting that payment processing is not yet configured.
- Decide whether the mobile app will be distributed in the EU, since Cyber Resilience Act reporting duties apply once it is on the EU market.
- Approve the privacy-policy wording that names the geocoding and breached-password services; changing the policy prompts users for consent again.

30. Developer Review Checklist

- Restore the hosted database, rebuild the live site against it, and confirm the production check and the live-services job pass.
- Add an explicit electronic-signature consent control to the mobile client.
- Add a check that every content-security-policy origin appears in the sub-processor list.
- Add a software bill of materials per release and rehearse the Cyber Resilience Act reporting runbook before any store release.
- Revisit the held major-version dependency updates once the web test suite exists.
- Apply the audit-log migration and authentication baseline to the live environment after the recorded pre-checks.
- Enable the scheduled automated production check.
- Provision a tier-1 environment, run the 10k load profile, and replace the first designed-for row with a measurement.
- Run a restore drill and replace the recovery-time targets with a measured figure.
- Add a unit and browser end-to-end suite for the web client.
- Correct and verify the payment integration before enabling it.
- Complete offline retry semantics and signature preservation across a failed replay.
- Record checklist identity and version, and require confirmation for fallback content.
- Perform physical-device testing on both platforms.
- Keep the configuration boundary and generator invariants authoritative.

31. Status Statement

EquipCert contains a coherent inspection application: a statically exported web client with an engineered interactive interface and measured accessibility, a purpose-built Flutter client for both mobile platforms, a tenant-aware schema built from versioned migrations, database-enforced authorization, private evidence storage, an unforgeable and erasure-compatible audit log, offline support, provider-independent and disclosed AI assistance with machine-readable content marking, consent and data-subject-request handling, versioned Terms of Service and privacy policy written against US and EU law, a published vulnerability-disclosure policy, layered and verified security controls, a horizontally scalable origin with generated orchestration, verified graceful-drain and health-probe behavior, published service objectives with runbooks, a automated production check, a live deployment proven identical to its built release, and a public documentation set covering law, security, architecture, and a phased roadmap.

What prevents an unqualified production-readiness claim is a short, specific list: a hosted database that is unreachable as of 24 September 2026, an unconfigured payment integration, legal documents awaiting counsel review, an unmeasured database tier, no measured availability period on a redundant tier, two live-environment changes awaiting application, no web user-journey suite, no physical-device or independent security verification, no explicit signing-consent control on mobile, and two third-party services still to be named in the privacy policy.

Update, 1 October 2026: three items on that list are closed — the database is reachable and the live site rebuilt and redeployed with checksum parity, the audit-log migration is applied and tested against the hosted database, and the mobile client has an explicit signing-consent control. Continuous integration passes in full. What remains: the unconfigured payment integration, counsel review, an unmeasured database tier, no measured availability period, the authentication-settings baseline, no web user-journey suite, no physical-device or independent security verification, the privacy-policy text for two third-party services, storing signing consent as its own evidence, and a separate test environment.

Every one of those is a known, scoped item rather than an unknown. The architecture for ten thousand users is built; the path toward one million is designed and documented; the evidence for each is labelled for exactly what it is.

This document is the handoff baseline for understanding the product exactly as verified in mid-September 2026 and updated on 24 September 2026.

Developed by Zahidul Islam

Inspection evidence, legal and regulatory engineering, layered security, scalable and available architecture, measured verification, and maintainable product handoff.