INDEPENDENT DUE-DILIGENCE ASSESSMENT

PPE1 Platform

Forensic Code Review

A file-level review of a five-solution mortgage-banking software platform — its architecture, code quality, data integrity, security posture, and, above all, whether an acquiring team can **own and extend it**.

PREPARED FOR

Prospective acquirers & licensees — lenders and SaaS providers

SUBJECT	DATE OF REVIEW	CLASSIFICATION
PPE1 — a Mortrax Corporation platform	June 27, 2026	Acquisition/licensing targets

1 — ENGAGEMENT & SCOPE

The mandate

Mortrax Corporation engaged ~~*[Company Name Redacted]*~~ to perform a thorough, forensic-level code review of five mortgage-banking domain software solutions comprising a platform referred to as **PPE1**.

The review was conducted at the source level across the complete codebase — every project, the shared database schema, the automated test suites, the build and continuous-integration configuration, and the platform's internal knowledge documentation. It was scoped to answer the questions a serious acquirer or source-code licensee asks first:

- **Is it real engineering?** Coherent architecture and consistent patterns, or assembled fragments.

- **Is the domain correct?** Does it model mortgage pricing and eligibility the way an industry veteran would.
- **Is it safe?** Authentication, authorization, injection resistance, data protection, dependency hygiene.
- **Is it ownable?** Can a new engineering team understand, maintain, and extend it *without* the original author.

These questions were operationalized as an explicit, comprehensive **code-review standard** — a detailed checklist of the properties verified at the file level across every solution, from authentication and multi-tenant isolation to injection resistance, data integrity, and enforced code policy. That full standard is enumerated in **Appendix A — Code Review Standards**, and it was applied uniformly across the codebase — so the findings here reflect the code *as it stands*, not a sampled spot-check.

---

2 — EXECUTIVE SUMMARY

Hand-built, with depth and discipline.

PPE1 is a modern, professionally architected mortgage program & pricing platform — built for two kinds of owner. A mid-size or large **lender** can deploy it in-house to price and manage its own loans; equally, a **SaaS provider** can offer it to lenders and brokers as a product — whether to launch as a new provider in the mortgage-technology space or to fold a polished program & pricing capability into an existing offering. The platform is the design of its **founder** — a mortgage-banking domain expert with 40 years in the industry, and the **fourth** mortgage-pricing technology he has designed since his groundbreaking, nationally-recognized first system in 1997. It is **hand-coded** software, and its depth, breadth, and internal consistency reflect a sustained, senior-level engineering effort. The central finding of this review is that the result reads as a **deliberately designed, domain-driven system** — coherent, disciplined, and built to be owned.

Across all five solutions the engineering is **consistent and intentional**: the same layered architecture (Controllers → business providers → data repositories → stored procedures), the same dependency-injection and error-handling patterns, the same naming and data-access conventions, and code organized by *domain responsibility* (a Program Engine, a Pricing Engine, a Bulk Pricer) rather than by technical layer alone. The design is **API-first**: the business logic lives behind clean services that every client — web and mobile — calls, so the platform is headless and integration-ready by construction. The data tier is governed by policy — effectively all access flows through roughly 490 parameterized stored procedures, soft-deletes preserve a full audit trail, and the schema enforces referential integrity with 96 foreign keys across 163 tables.

The platform is current and cohesive: **.NET 10** across the server tier — ASP.NET Core MVC with Razor views for the administrative hub, and Web APIs for the services — with a **React 18 + TypeScript + Vite**

client for the pricing tool — delivered as both a web and a **mobile** application from one shared core — plus source-generated object mapping and a clean dependency profile (the front-end reports zero `npm audit` vulnerabilities). Performance is strong and measured — the engine prices a 250-loan batch in under 49 seconds and returns full interactive quotes in under two seconds per loan (see Section 3). Every solution carries an automated test suite wired to continuous integration — over 1,700 tests across the platform, all green. The platform is also self-documenting to an unusual degree, including a knowledge product ("eLCee") that generates living documentation directly from the source of truth.

> **Bottom line up front.** PPE1 presents as a maintainable, coherent, and transferable asset. Its design choices — stored-procedure data access, soft-delete discipline, a shared-core monorepo, enforced nullable-reference and dead-code policies, and deep documentation — are precisely the choices that make a codebase *ownable* by a team that did not write it. The answer to "can our engineers take this over and extend it?" is, on the evidence, **yes**.

---

3 — THE PLATFORM AT A GLANCE

Five solutions, one platform

PPE1 is delivered as five independently-versioned solutions that share a single database and a single JWT identity. Users authenticate once through **Identity**, work in the **Manager** hub, and reach the **Loan Pricer** tool — backed by the pure-logic pricing API — with the **Learning Center** available as a living-documentation companion.

	Identity Authentication	**Manager** Administrative hub	**Loan Pricer API** Program & Pricing engines	**Loan Pricer UI** React · Web & Mobile	**Learning Center** AI knowledge
Technology stack	.NET 10 ASP.NET Core Web API · Dapper + stored procedures · JWT (HMAC-SHA256)	.NET 10 ASP.NET Core MVC + Web API · Razor + jQuery · Dapper/SP · Mapperly	.NET 10 ASP.NET Core Web API · Dapper/SP · dual-engine · dynamic-rule evaluation	React 18 · TypeScript · Vite · Material UI · Redux Toolkit / RTK Query — **web + mobile** clients from one shared core	Node + TypeScript (Express + AI SDK) engine · React + Vite app
Lines of code	2,829	168,255	29,885	40,827	4,761
Source files	59	705	292	210	49
Platform total	246,557 lines of hand-written code across 1,315 source files (vendor libraries, reference-only code, and CI tests excluded) · published publicly as **240K+ LOC / 1,300+ files**				

Counts produced by the platform's deterministic line-counting tool on the date of review; they exclude vendor and build artifacts, and CI tests.

SHARED DATA TIER & QUALITY SIGNALS

493	163	0	5 / 5
Stored procedures (the data layer, by policy)	Database tables (96 foreign keys, by design)	Front-end dependency vulnerabilities (npm audit)	Solutions with green CI test suites

Schema also includes 59 functions and 25 views. Data access is by policy **stored-procedure based**; the few inline statements that exist are a deliberate, reviewed exception (a handful of self-service profile toggles), not an ad-hoc practice.

MEASURED PERFORMANCE

<49s	<0.2s	<2s	1000s
to price a 250-loan Bulk Pricer batch	per loan in batch (parallelized)	per loan, full-scope interactive pricing	of loans per batch (institutional scale)

Performance was measured on representative workloads. The **Bulk Pricer** priced a random sample of **250 loans in under 49 seconds** — **under 0.2 seconds per loan** — using thread-safe parallel processing, and is built to handle thousands of loans per batch at institutional scale. The interactive **Program and Pricing engine**, which evaluates a far larger scope of rates and prices per request, returns complete best-execution results in **under two seconds per loan**. These figures are verifiable in the platform's processing history and make the architectural point directly: the engine's depth does not come at the cost of speed.

---

4 — SOLUTION-BY-SOLUTION ANALYSIS

A closer look at each product

4.1 Identity — The authentication authority

.NET 10 Web API · Dapper / stored procedures · JWT · 2FA · 243 tests

Identity is the single authentication authority every other component trusts. A user signs in once and the resulting JSON Web Token is honored across the Manager hub and the Loan Pricer tool — platform-wide single sign-on resting on one shared signing key, issuer, and audience. It is the sole home for anything that touches a username or password.

NOTABLE ENGINEERING

- **Stateless two-factor authentication.** A short-lived, signed pre-auth token carries the "password validated, awaiting code" state between login steps — so 2FA works across web and mobile with no server-side session store. One-time codes are hashed at rest, with automatic SMS-to-email fallback.
- **Per-session refresh-token rotation.** Each session holds its own refresh token; refreshing rotates only the presented token, so concurrent devices never invalidate one another — a clean, secure model.
- **Defense-in-depth account lifecycle.** Self-service registration, time-boxed activation, single-use password reset, and an anti-phishing "company challenge" — every account-discovery surface is anti-enumeration, tokens are cryptographically random and hashed, and abuse-prone endpoints are rate-limited.
- **Clean four-layer separation** (contracts → data access → services → API host), stored-procedure data access throughout, and signed authorization claims (defense-in-depth, not client-supplied).

4.2 Manager

The administrative hub

.NET 10 MVC + Web API | Razor + jQuery | Mapperly | 580 tests

Manager is the platform's main application — the console where mortgage professionals configure and manage the core data, and provides web access to the Loan Pricer tool. It is the largest solution by far, and the breadth is organized, not sprawling: every project is structured by business module — company management (clients, brokers, investors, mortgage insurers), program management (programs, products, guidelines, pricing adjustments, SRPs, and overlays), rate-sheet management with spreadsheet migration, and utility imports of government and market data.

NOTABLE ENGINEERING

- **Uniform layered architecture** applied module-by-module across eight projects — controller → provider (with source-generated Mapperly mapping) → repository → stored procedure — with a clean three-tier type split (entities, transport DTOs, view models).
- **A hardened embedding boundary.** The Loan Pricer React app (web version) is embedded within Manager as a deliberate security boundary: Manager resolves the user's capabilities to plain boolean flags *server-side* and never lets internal role identifiers cross into the embedded app, passing only a server-rendered token, the resolved flags, and the theme.

- **Disciplined data access & integrations.** Stored-procedure-only access with soft-deletes throughout; live integrations with FRED, HUD, FHFA, FFIEC, the Census geocoder, and Freddie Mac for the import utilities; and proactive supply-chain hygiene (vulnerable transitive packages pinned up with documented rationale).

4.3 Loan Pricer API

The Program / Pricing engine — the "brain"

.NET 10 Web API | Dual engine | Bulk Pricer | 498 tests

This is the heart of the platform: a pure business-logic API with no UI that decides which investor programs a loan qualifies for and at what price. **Both Loan Pricer access points — the full-featured web client and the mobile client — call this one engine**, so identical business logic drives every quote regardless of how the user arrives. It is the deepest and most domain-rich solution, and its design is its strongest argument that PPE1 is the work of a genuine practitioner.

TWO COOPERATING ENGINES

- **Program Engine — eligibility.** A six-stage chained-executor pipeline (a clean Chain-of-Responsibility) evaluates a loan against guideline data and program overlays, including a **credit analysis** that derives the borrower's credit grade — a determination that flows through to both eligibility and pricing. Overlay rules apply with explicit cap / adjust / kill semantics — they can only ever *restrict*, never improve. When a loan narrowly misses, a depth-guarded **counter-offer cascade** automatically searches for the maximum-qualifying terms; combination (first + second lien) loans are handled as paired pipelines; and every stage is logged for full audit and replay.
- **Pricing Engine — price.** A fixed, auditable sequence — evaluate rate-sheet products against the rules; accumulate overlay / LLPA / SRP adjustments; resolve the loan's **ARM margin and rate caps** and its **prepayment** terms; apply all of it to base pricing; then rank by best execution — with user vs. admin quote modes.
- **Bulk Pricer.** Institutional-scale batch pricing of thousands of loans from a spreadsheet, with configurable thread-safe parallelism, address geocoding, reprocessing and rollback, real-time distribution dashboards, and a drag-and-drop bid-tape builder.

NOTABLE ENGINEERING

- Free-form business rules (authored in Manager) are evaluated through a dynamic-expression layer with a self-invalidating compiled-expression cache — fast, and driven by data rather than hard-coded.
- **Enums generated from the database** (the database is the single source of truth), stored-procedure-first access with a consistent `Lp` prefix, and an **audit-by-replay** design where every engine stage is logged and can be re-executed.

- Golden-master **characterization tests** guard the engine and mappers against regression — exactly the right test discipline for a high-stakes calculation core.

> The domain logic here encodes edge cases only an industry veteran would know — VA entitlement handling, mortgage-insurance irrelevance at or below 80% LTV, high-balance county classification, and the forced counter-offer cascade. This is the difference between software that *looks* like a pricing engine and one that a lender can actually price loans on.

4.4 Loan Pricer UI

React · web + mobile clients

React 18 + TypeScript | Vite · Material UI | RTK Query | 269 tests · 0 vulns

The "face" of the Loan Pricer, this is a modern React + TypeScript application that holds no business logic — every pricing, guideline, and lookup operation is a call to the API. It is built as an **npm-workspaces monorepo** whose shared, UI-agnostic core package (types, data services, hooks, contexts, theme tokens) is the foundation for **two first-class clients — web and mobile** (Section 4.5): the platform's business logic, data layer, and validation are written once and run on both. As the desktop access point delivered inside the Manager hub, the web client carries the most extensive feature set — interactive pricing, the loan **Pipeline**, the institutional **Bulk Pricer**, and administrative **Tools** — all driven by the same Loan Pricer API.

NOTABLE ENGINEERING

- **Clean separation of concerns** — zero business logic in the client; one data-access slice per domain over a shared, resilient request layer (bearer-token auth, per-request timeouts, friendly error handling, silent token refresh on mobile).
- **Production-grade UX engineering** — virtualized data grids that stay responsive at 10,000+ rows, a full dark-mode system, and a carefully managed embedding handshake (token / theme / session keep-alive) so the embedded experience feels native.
- **Current, clean toolchain** — React 18, TypeScript in strict mode, Vite, and a Vitest suite of 263 tests; the dependency tree reports **zero** known vulnerabilities.

4.5 Mobile

The mobile client · direct access

React + TypeScript · Vite | Native · iOS + Android | Shared core | Play Store + App Store underway

The platform reaches the phone through a complete, dedicated **mobile client** — a **direct access point** that reaches loan officers in the field without routing through the Manager hub — built on the very same shared core as the web pricing tool, the architectural foresight of the monorepo realized. Business logic, data services, and validation are written once and run on both faces; the mobile app contributes only its own screens and a phone-native presentation.

NOTABLE ENGINEERING

- **The full workflow, in hand.** Mobile delivers the complete authenticated experience — two-factor sign-in, account self-service, loan entry, best-execution pricing, loan audit, and counter-offers — adapted to a single-hand, phone-native interface while tracking the web product's look and behavior for one consistent platform.
- **Write-once architecture, proven.** Because both clients consume the same shared package, a change to a pricing rule, a data service, or a validation rule runs identically on web and mobile — the strongest possible evidence that the shared-core design was deliberate, not incidental.
- **One backend, multiple faces.** The mobile client authenticates directly through Identity and prices through the same Loan Pricer API as the web tool — one business-logic core, two access points. It is packaged as a **native mobile app** for iOS and Android — built from the shared React core, with native device features — and its release to the **Apple App Store** and **Google Play** is underway.

> Mobile is not a port or an afterthought — it is a full, first-class client in its own right, one the platform's architecture was built to support. Bringing a pricing platform of this depth to loan officers in the field — on a phone, from a single shared codebase — is a meaningful strategic and competitive asset.

4.6 Learning Center — "eLCee"

The AI knowledge product

Node + TypeScript | React + Vite | Agentic RAG | 136 tests · eval-harnessed

The Learning Center is a buyer-facing differentiator: an AI knowledge product, persona "eLCee," that answers questions about how the platform works by drawing on a curated corpus written against the live codebase. Its value proposition speaks directly to acquirer concern —

living documentation generated from the source of truth, which cannot go stale because it is grounded in the real system.

NOTABLE ENGINEERING

- **An agentic answer pipeline** (a TypeScript engine over an AI provider's official SDK) with streaming responses, identity-tiered retrieval, and multi-turn context — wrapped in a **deterministic, layered safety guard** that does not rely on model compliance for its guarantees.
- **Engineered for trust:** access content is derived server-side from the authenticated user's role; a hard floor of protected material is refused for everyone; and the system is validated by an automated evaluation and adversarial-guard harness run in continuous integration.
- A reserved, documented .NET tier stands ready for the future durable-state layer — a deliberate architectural placeholder, not abandoned scaffolding.

---

5 — CROSS-CUTTING ASSESSMENT

How the platform holds together

API-first by design

Every piece of business logic lives behind a clean API: the Loan Pricer engine is a pure, UI-less service, and every client — web and mobile — calls it for all pricing, eligibility, and data. Nothing of consequence is trapped inside a screen. That is what makes the platform genuinely **headless and integration-ready** — a new front end, a partner system, or an embedded use case can be built against the same APIs without touching the core.

Multi-tenant by construction

A single deployment hosts many independent lending companies — each with its own organizational hierarchy (region → branch → loan officer) and an optional wholesale division (broker company → branch → account executive) — with per-client data-ownership scoping enforced platform-wide over a privileged system-owner tenant. This is the tenancy model a SaaS provider needs, already built and exercised — so the dual lender-or-provider proposition rests on real machinery, not a slide.

Auditable by design

PPE1 is engineered to be *interrogated*, not merely trusted. For any loan — even one priced months ago — the engine can **replay** its evaluation field by field and attribute every outcome to a specific guideline, an overlay rule (with the condition that fired it), or the input data — answering the hardest question a pricing platform faces, *"is it our code or their data?"*, as a defensible, repeatable result. Access decisions are likewise explainable and self-checked, authentication is fully logged, and because nothing is ever hard-deleted, every change is reversible and the full history browsable. Observability and governance of this depth materially de-risk owning a calculation engine.

Architecture & consistency

One layered model — controller → provider → repository → stored procedure — applied uniformly across every .NET solution, with interface-segregated dependency injection throughout. Patterns repeat by design; there are no one-off architectures hiding in corners.

Configured, not coded

Eligibility, guidelines, and price adjustments are authored as **data** — through one shared visual rule system over a single field catalog, with guideline inheritance so a large investor-program catalog is maintained once and specialized where needed. A licensee can reshape behavior without touching source: the extensibility an in-house team or SaaS provider needs to make the platform its own.

Real-world data engineering

Eligibility and pricing are fed directly from the official primary sources — FHFA, HUD, FFIEC, Freddie Mac, Census, the Federal Reserve — with no third-party data reseller in between, while investor ratesheets are normalized into base pricing at a scale of over a million records a day. Batch pricing is resilient by design (a single malformed row is reported, never fatal) and produces investor-ready deliverables. This is built for the messy reality of the secondary market, not a demo.

Data access & integrity

Stored-procedure-based access (minimal inline SQL, by policy) gives effectively no SQL-injection surface in application code. Soft-deletes preserve a full audit trail, system-versioned temporal tables retain loan history, and 96 foreign keys enforce referential integrity across the schema.

Performance & scale

Pricing scales on the compute side without contention: batch and bulk runs execute on thread-safe parallel paths — a fresh executor chain per loan, isolated per-item state, pooled connections, and thread-safe counters — so throughput grows with available cores rather than colliding. Hot data paths evaluate only the records that can actually apply, virtualized grids are fed by correct server-side paging with no unbounded queries, and bulk or test execution is isolated from production data so a large run never touches live records. Measured throughput meets its stated targets.

Test & CI posture

Every solution carries an automated test suite — golden-master characterization on the calculation core, pure-logic unit tests on the services, and component tests on the front ends — each wired to a continuous-integration workflow with a passing status check — **over 1,700 tests in all**, spanning unit, integration, and end-to-end layers.

Modern, clean toolchain

.NET 10, React 18, TypeScript, Vite, and source-generated mapping (no reflection-based mapper, no associated CVE surface). The front-end dependency tree reports zero known vulnerabilities, and vulnerable transitive packages elsewhere are explicitly pinned up.

Documentation depth

An extensive internal knowledge base captures business rules, architectural decisions, and data-model rationale — and the Learning Center turns the source of truth itself into queryable, self-refreshing documentation. This is materially above what most human-only teams maintain.

---

6 — DEPTH OF ENGINEERING

The scope this represents

A platform of this depth, breadth, and internal consistency represents a substantial body of senior-level engineering — the kind of system that takes an experienced team significant time to design, build, and harden. Several characteristics make that effort evident in the code itself:

- **Domain accuracy.** Every business rule reflects decisions a 40-year mortgage-banking expert would make. The system handles edge cases — VA no-limit entitlement, MI irrelevance at ≤80% LTV, high-balance classification, the forced counter-offer cascade — that only deep, hands-on industry experience produces.
- **Architectural consistency.** The same dependency-injection, repository, error-handling, and naming patterns recur across all five solutions — a single coherent design carried through tens of thousands of lines, not stitched-together fragments.

- **Domain-driven organization.** Code is grouped by business responsibility (Program Engine, Pricing Engine, Bulk Pricer, Shared), not by technical layer alone — design-level thinking expressed in the folder structure.
- **Sophisticated data modeling.** Deliberate schema decisions — a single discriminated section table, a unified inheritance mechanism, provider-based business logic over ad-hoc handlers — reflect considered design engineering rather than incidental structure.
- **Enforced policy.** Nullable reference types, stored-procedure data access, soft-delete patterns, dead-code removal, and stored-procedure formatting standards are *enforced policies*, not aspirations; the codebase compiles with tens of warnings, not hundreds.

---

7 — OBSERVATIONS & TRAJECTORY

Honest about the trade-offs

No platform of this scope is without areas of continued investment, and a credible review names them. In PPE1's case they are **known, documented, and actively managed** rather than hidden:

- **Test coverage is broad and still widening.** Over 1,700 CI-gated tests run green across the platform — golden-master characterization on the calculation core, plus unit, integration, and end-to-end layers; the end-to-end layer is the newest and is the one still being extended by risk priority.
- **Disciplined technical-debt management.** The team maintains an explicit, prioritized technical-debt backlog and a documentation-freshness discipline — the platform's modernization items are tracked and sequenced, not accumulating silently.

The presence of these items, openly tracked, is itself a maturity signal: this is a platform engineered with the long view, by a team that distinguishes shipped from in-flight.

---

8 — SECURITY & DATA PROTECTION

Built to be trusted with a lender's most sensitive data

A pricing platform holds a lender's and a borrower's most sensitive data, so security and data protection were verified at the file level across every solution. Six dimensions:

- **Authentication & session.** A single signed-JWT authority guards every protected surface with no bypass path; authorization claims are derived server-side and signed, never trusted from the client; two-factor state rides a short-lived pre-auth token with codes hashed at rest; refresh tokens rotate per session; and account-discovery endpoints are rate-limited and hardened against enumeration.
- **Multi-tenant isolation.** Every data-returning path scopes its results to the caller's client, resolved from the verified token on the server — closing cross-tenant leakage on both client- and id-named parameters, over an explicit system-owner boundary.

- **Injection & input safety.** Effectively all data access flows through parameterized stored procedures (minimal inline SQL, by policy), leaving a negligible injection surface in application code; external input binds to typed models with enforced required fields, so malformed input yields a clean `4xx` rather than a `500` or a silent default.
- **Secrets & supply chain.** No secrets, connection strings, or signing keys live in source or history — configuration flows through per-environment stores; source-generated mapping leaves no reflection-mapper CVE surface; and the front-end dependency tree reports zero known vulnerabilities.
- **Borrower-data protection.** PII is handled on a need-to-know basis, credentials are hashed at rest, and traffic is served over TLS end to end; secrets and PII are scrubbed from logs and from the AI knowledge corpus — sensitive values never reach a log sink or a model prompt.
- **Auditability.** Authentication events are logged, access decisions are explainable and self-checked, and because nothing is hard-deleted, every change is reversible and the full history remains browsable.

9 — CONCLUSION

A coherent, ownable, domain-true platform

PPE1 is a professionally architected, domain-driven mortgage-banking platform whose code quality, consistency, and documentation reflect a substantial and sustained engineering effort. Its data layer is governed by policy, its schema enforces integrity by design, its stack is current and clean, and every solution ships with automated tests gated by continuous integration.

Most importantly for any acquirer — a **lender** taking it in-house or a **SaaS provider** taking it to market — the platform is built to be **handed off**. The uniform layered architecture, the shared-core monorepo, the stored-procedure data discipline, the enforced code policies, and the self-refreshing living documentation are exactly the properties that let a new engineering team take ownership and extend the system with confidence — and the deep domain logic is the kind that would take years, hugely significant budget provisions, and the sustained involvement of a scarce senior mortgage-domain expert to rebuild from scratch.

On the strength of this review, qualified parties are encouraged to proceed to the full data room and to interrogate the platform live through the Learning Center.

Assessment: an enterprise-class, transferable, domain-true asset.

APPENDIX A

Code Review Standards — what this review examined

This review was conducted against an explicit, **comprehensive standard**: the codebase was examined at the file level across every solution, and each property below is something the review actively verified — not a general aspiration. Together these criteria form the complete standard the code was measured against; the findings throughout this document are the *result* of applying it.

A1 Authentication & session

- ✓ A single authentication authority; every protected surface validates a signed token, with no bypass path.
- ✓ Authorization claims are **server-derived and signed** — never read from a client-supplied or client-writable source (e.g. a cache the client can set).
- ✓ Two-factor state is carried in a short-lived, signed pre-auth token; one-time codes are hashed at rest with a safe delivery fallback.
- ✓ Per-session refresh-token rotation; concurrent devices are isolated; refresh rotates only the presented token.
- ✓ Idle/session timeout enforced; token lifetimes bounded; sign-out invalidates cleanly.

A2 Authorization & multi-tenant isolation

- ✓ Every data-returning endpoint scopes results to the caller's client/tenant — verified for cross-tenant leakage on both `clientId`- and `id`-named parameters.
- ✓ Tenant identity is resolved from the **verified token on the server**, by a fresh read — never from a cache that could fail open to a privileged tenant.
- ✓ Role checks resolve by a **stable role identity**, not a mutable display name or an environment-specific database id.
- ✓ The system-owner tenant boundary is explicit; privilege-escalation and cross-silo paths are closed.
- ✓ The Manager → embedded-app boundary passes only resolved capability flags and a server-rendered token — internal role identifiers never cross it.
- ✓ Cross-origin / embedding origins are restricted to an explicit environment allow-list.

A3 Injection, input & output safety

- ✓ All data access is parameterized (stored procedures / parameterized commands); no string-built SQL in application code.
- ✓ External input is bound to typed models with enforced required fields; malformed input yields a clean `4xx`, never a `500` or a silent default.
- ✓ No untrusted value reaches a shell, file path, or dynamically-built query unescaped.
- ✓ Output crossing a trust boundary (e.g. the AI knowledge corpus) is **deterministically scrubbed**, not left to model compliance.

A4 Secrets, dependencies & supply chain

- ✓ No secrets, connection strings, or signing keys are hardcoded in source or committed to history; configuration flows through environment / secret stores.
- ✓ Dependency trees carry no known vulnerabilities; vulnerable transitive packages are explicitly pinned up with documented rationale.
- ✓ **Source-generated** mapping leaves no reflection-mapper CVE surface.
- ✓ Build and CI configuration are reviewed for credential exposure.

A5 Concurrency, performance & scale

- ✓ Batch and parallel paths are thread-safe: isolated per-item state, a fresh executor chain per item, pooled connections, thread-safe counters.
- ✓ Bulk / test execution is **isolated from production data** — no side-effect writes to the live schema.
- ✓ Hot data paths evaluate only the records that can apply; measured throughput meets the stated targets.
- ✓ Virtualized grids are fed by correct server-side paging — no unbounded queries.

A6 Testing, CI & observability

- ✓ Every solution carries an automated suite wired to continuous integration with a passing gate.
- ✓ **Golden-master characterization** guards the calculation core against regression.
- ✓ Integration tests exercise real stored procedures and endpoints (transaction-rolled-back), not only mocks; a new SP or endpoint requires a matching test.
- ✓ Authentication events are logged, access decisions are explainable, and because nothing is hard-deleted the full history is browsable.

A7 Architecture & maintainability

- ☑ One uniform layered model — controller → provider → repository → stored procedure — applied consistently, with no one-off architectures hiding in corners.
- ☑ Interface-segregated dependency injection; no hidden static or global coupling.
- ☑ Code is organized by **domain responsibility**, with clear seams and extension points a new team can build on.
- ☑ **API-first**: business logic lives behind services; nothing of consequence is trapped inside a screen.

A8 Documentation & ownability

- ☑ Business rules, architectural decisions, and data-model rationale are captured in a maintained knowledge base.
- ☑ Living, source-grounded documentation that cannot drift from the code (the Learning Center reads the real system).
- ☑ A prospective owner can trace any behavior from UI → API → data **without the original author**.

A9 Data integrity & persistence

- ☑ Referential integrity is enforced by foreign keys; relationships are not orphan-prone.
- ☑ **Soft-delete discipline** throughout — no hard deletes; active-record filtering applied consistently in every read path.
- ☑ System-versioned temporal history is correct: right period columns, correct delete semantics, no unintended version bloat.
- ☑ A retrieved record shows its **accurate stored values or is blanked and blocked** — never silently substituted to a default or first available option.
- ☑ Stored-procedure objects compile with correct `QUOTED_IDENTIFIER` / `ANSI_NULLS` settings (no latent XML-path failure) and retain their multi-line definitions.

A10 Correctness & domain logic

- ☑ Every business rule matches mortgage-domain ground truth — VA entitlement, MI irrelevance at ≤80% LTV, high-balance county classification, the counter-offer cascade, ARM margin/caps, and prepayment terms.
- ☑ Engine decision points are logged and **reproducible via replay**; a result can be attributed to a guideline, an overlay rule, or the input data.
- ☑ Overlay and adjustment semantics are correct and one-directional (restrict-only — they can never improve an outcome).
- ☑ Edge cases are handled explicitly: missing credit score, combination (1st+2nd) loans, wholesale scope, zero/degenerate inputs.

A11 Code quality & enforced policy

- ☑ Nullable reference types are honored; null-flow warnings are resolved, not blanket-suppressed.
- ☑ No dead code — unreachable methods, orphaned endpoints, and unreferenced stored procedures are removed across the **full call chain**.
- ☑ Exceptions are rethrown preserving stack traces; no silent `catch` masks a failure as a benign result.
- ☑ `async` only where awaited; no sync-over-async.
- ☑ Names describe **purpose** — no legacy/heritage or defect terminology in shipping code; filename casing is stable (git case-sensitivity respected).

A12 Build, configuration & operability

- ☑ The platform builds cleanly from source with documented steps — no undocumented local dependencies or hidden manual stages.
- ☑ Configuration is **environment-based** (dev / test / prod); nothing environment-specific is baked into code, and secrets are externalized to per-environment stores.
- ☑ Database schema and reference data are **scripted and repeatable** — an environment is reproduced, not hand-tuned.
- ☑ Deployments are documented and reversible; a known-good path exists to stand the platform up or roll it back.
- ☑ A new owner can bring up the entire platform from the repository and data room — the operational knowledge is **captured, not tribal**.

A13 Privacy & data protection

- ✓ Borrower PII is handled on a need-to-know basis — sensitive fields are not exposed beyond the endpoints that require them.
- ✓ Credentials are **hashed at rest**; secrets and connection strings live in per-environment stores, never in source or history.
- ✓ Traffic is served over **TLS end to end**; internal service-to-service calls carry the signed token, not raw credentials.
- ✓ Logs and diagnostics are scrubbed of secrets and PII — sensitive values never reach a log sink or the AI knowledge corpus.

A14 Resilience & fail-safe behavior

- ✓ Writes are **transactional** — an operation either completes or rolls back cleanly; no partial or orphaned state.
- ✓ Authorization and session checks **fail closed** (deny), never open, when a token, cache, or dependency is unavailable.
- ✓ Idle/session-timeout state stays consistent between client and server — an active session is never silently dropped, nor a signed-out one silently honored.
- ✓ External and cross-service calls are bounded by timeouts; a degraded dependency surfaces a clean error rather than a hang.

A15 Front-end integrity & UX consistency

- ✓ Nothing of consequence lives in the screen — the UI calls services for every decision, reinforcing the API-first boundary.
- ✓ The UI shows a record's **true stored state, or blanks and blocks** — never a silently substituted default or first-available option.
- ✓ One shared design system and light/dark theming applied consistently; the mobile client tracks the desktop app's layout and labels.
- ✓ Client input is validated for a good experience but **never trusted for authority** — the server re-validates everything.

A16 API contract & compatibility

- ✓ Endpoints expose stable, typed request/response contracts bound to explicit models.
- ✓ Model and nullability changes **cannot silently break a client** — required-ness is explicit and intentional, with no implicit-required regressions.
- ✓ Errors cross the boundary as consistent typed responses (a clean `4xx`), never a `500` or a silent default.
- ✓ One contract is consumed identically by the web app, the mobile client, and any external integrator.

As applicable, each criterion above was verified across all five solutions in the code as it stood on the date of review. Together they are the full standard this assessment applied — the coverage guarantee behind its conclusions.
