AegisVeritas: Technical Report on the Final Product

Samuel Mwania, BSc Software Engineering (Machine Learning), African Leadership University. Supervisor: Kevin Sebineza.
October 2026.
Approved proposal: *Uncovering Hidden Supplier Links in Kenyan Public Procurement: A Graph-Based Machine Learning Approach* (23 September 2026). Repository: github.com/mwaniasam/aegisveritas. Walkthrough video: youtu.be/g4tlftDK7T4.

AegisVeritas finds bidders in Kenyan public procurement that are linked to debarred parties, or to each other, in ways a name check misses. It reads public award, ownership and contact records, screens bidders against sanctions and debarment lists, follows ownership paths, and ranks supplier pairs by how likely they are to share an owner. Every result shows its evidence and is written to a tamper-evident log. A person always makes the decision (Data Protection Act 2019, s. 35).

1. Scope alignment and feature validation

Each feature planned in the proposal (objectives 1 to 6), its purpose and its status. The code for each is in the repository.

Planned feature	Purpose	Status	Evidence
Review of existing mechanisms (Obj. 1)	Show why name checks miss hidden links	**Completed**	Proposal ch. 2, approved 23 Sep 2026
Graph from six public sources (Obj. 2)	One record of firms, people, awards and adverse records	**Completed**	PPRA/PPIP, World Bank, OpenSanctions, ICIJ, EACC; one-command pipeline
Entity resolution, checked on 200 pairs (Obj. 2)	Merge records of the same party; measure errors	**Partial**	Resolution built; the 200-pair check moves to the dissertation
Graph learning vs. baselines (Obj. 3)	RQ3: do graph methods work on real Kenyan data?	**Completed**	Answer: no. On 18 real cases no model beat a size-only baseline
Bidder screening (Obj. 4)	Flag direct and hidden exposure	**Completed**	List match, exposure paths, model or rules; tested
Supplier-list screening (Obj. 4)	Screen a whole list at once	**Completed**	Batch page and service; tested
Ownership path tracing (Obj. 4)	Show the path to the listed party	**Completed**	Investigation canvas
Explanations (Obj. 4)	Every score states its reasons	**Completed**	Risk factors; evidence card on every pair
Audit trail (Obj. 4)	Accountability for every action	**Completed**	SHA-256 hash chain, checked in the app; decisions need a reason
Label isolation	Stop the model reading the answer	**Completed**	Answer-revealing links removed before scoring; tested
Scheduled re-screening	Re-check bidders when lists change	**Completed**	Celery worker with Redis; tested
Cloud deployment	A hosted system for the interviews	**Planned**	My decision: results name firms not found guilty of anything, so the system is not on the open internet. Runs in Docker Compose; next is invite-only access (`docs/DEPLOY_PRIVATE.md`)
Expert interviews, 12 to 15 (Obj. 5)	RQ4: accuracy, clarity, fit with daily work	**Not started**	Ethics approval 14 Sep 2026 (SEPT26-BEL-070); no interviews held yet; recruitment starts in November, before the dissertation

When the approach works (Obj. 6)	State the conditions and why	**In progress**	Six pre-registered studies (section 3)

Scope change. The proposal used debarred firms as labels. After data errors were fixed, only 18 confirmed Kenyan cases remained, too few to learn from. After the supervision meeting of 10 October, I moved the machine learning to a problem with far more labels: **related bidders**, pairs of suppliers that share an owner, labelled from the ownership details Kenyan awards must disclose (12,441 links). I added three features: the **evidence engine**, the **review queue with evidence cards**, and a **pre-award tender check**. Kenya has no open ownership register to test against, so I also validated the method in Estonia, where both are public.

2. Technical functionality

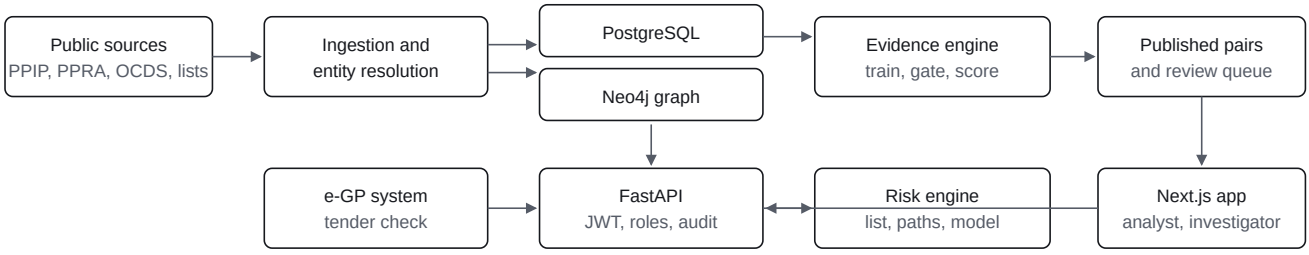


Figure 1: Components and data flow. The API reads published results; the engine runs as a separate scheduled job.

Screening a bidder. The API matches the name to one entity and reads its two-step neighbourhood from Neo4j. It then runs three checks in order: a direct list match; the shortest path to any listed party (breadth-first search that skips buyers, so a large ministry does not link every supplier); and the graph model, or simple rules when the firm has too few links. The audit entry is saved before the result is returned.

Scoring supplier pairs. Candidate pairs are firms that win from the same buyer within 30 days, or that share a disclosed record. Each pair gets three pieces of evidence: *identity* (shared owner or director), *contacts* (shared phone, email or address, worth less when many firms share it) and *behaviour* (LightGBM on 20 award-pattern features, trained on Kenyan and Estonian pairs). The behaviour part stays silent when a firm has fewer than 3 awards. The three are combined into one probability (Fellegi-Sunter). A new model goes live only if it beats the baseline and the current model on a fixed holdout.

Frontend and backend. Next.js pages call FastAPI through a typed client. All errors share one format, so every page shows a clear message. Only investigator, auditor and admin roles see named pairs.

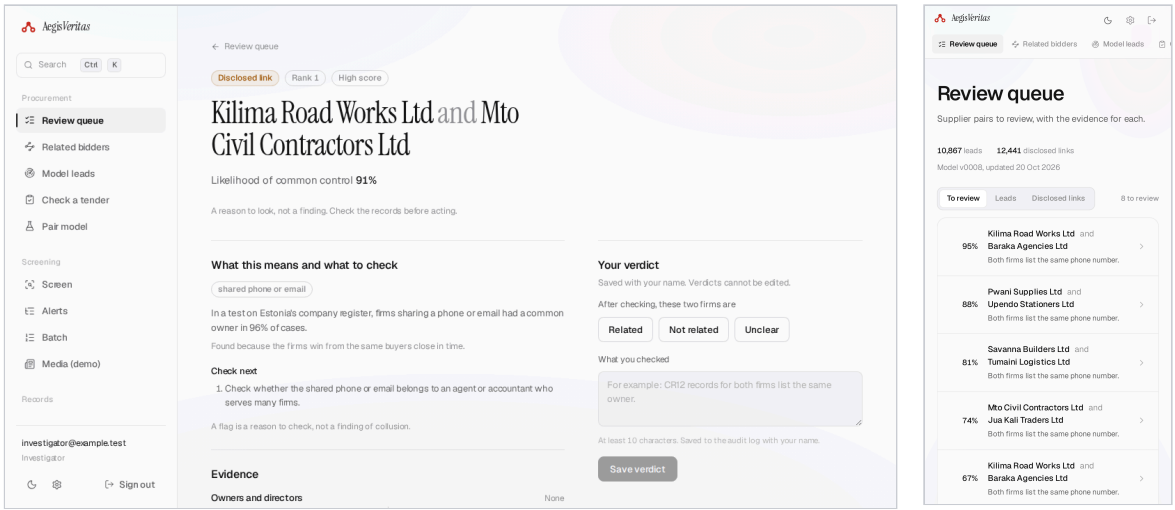


Figure 2: A pair with its evidence card and the verdict form (laptop), and the review queue (phone, 412 px). Firm names are invented; captured by the browser tests.

3. Testing approaches and results

Approach	Why and how	Result	What it changed
Unit and integration (pytest)	Catch silent breaks. Small cases per part; full engine cycles on a synthetic market	664 passed, 14 skipped (need a live database); 49% line coverage overall; 81% to 97% in the engine, screening and service code	A cap had cut published links from 12,441 to 5,000; fixed, with a test
Browser end-to-end (Playwright)	Pages must handle every API answer, on laptop and phone. 16 scenarios, incl. errors and a dead server	32 of 32 pass (desktop and Pixel 7). On my laptop, 80 of 80 also pass in Firefox and WebKit (Safari's engine), incl. iPhone 14	Errors now show the backend's exact message; wrong roles see a refusal, never data
System test	Call every endpoint and open every page of the running stack	54 of 54 checks pass with real data (first run: 52 passed, 2 warnings)	Unset Google/Microsoft sign-in returned 500; now 501 with a clear message
Pre-registered validation	Avoid fooling myself: each pass rule was on GitHub before its run	Estonia AUC 0.85 vs 0.57; top-1,000 recall 4% to 32%; Kenya AP 0.146 vs 0.067 (all passed). KeRRA register check: owner-linked pairs shared a P.O. Box 209.9 times more often than matched controls (30 of 253 vs 5 of 8,852, p = 1.0e-42; verification report section 5, 28 Oct). Estonia-only transfer and the KeRRA primary test failed, and are reported	The live model scored 0.075 because of untested changes; engine-3 now runs exactly the tested model
Performance and load	Must work on modest hardware. Page loads normal and on a slow phone; API alone and with 10 clients	Tables below	Capped chains and leads keep the list at 12,641 pairs

Page (median of 3)	Fast machine	Low-end phone
Landing page	212 ms, 1,129 kB	5,292 ms, 945 kB
Sign in	164 ms, 785 kB	1,768 ms, 785 kB
Review queue	208 ms, 512 kB	2,584 ms, 453 kB
Pair and evidence card	268 ms, 479 kB	2,780 ms, 442 kB
Tender check	248 ms, 487 kB	2,316 ms, 432 kB

Table: page load in ms to largest paint, and data transferred. Chromium 141; slow phone = 4x slower CPU, 1.6 Mbit/s, 412 px screen.

Workload	Result
API on real data (30 calls each)	Median 1 to 7 ms
Tender check, 2 bidders	Median 70 ms
Screening a company	Median 36 ms (first call 2.1 s)
10 clients at once, 20 s	155 requests/s, 0 failed
Full retrain and scoring, 1.44 million pairs	3 min 45 s, 2.6 GB; AP 0.146 again
Stress: 9,000 firms, 60 buyers	Ran out of 7 GB memory (next: batches)

Table: backend timings. On my laptop (Core Ultra 7, 15 GB) and a 2-core test machine.

Compatibility. All browser tests pass in Chromium, Firefox and WebKit, on desktop and phone sizes. On the slow phone, the working pages load in 2.3 to 2.8 s; the landing page takes 5.3 s because of its 3D scene, which I will lazy-load next.

4. Code quality, object-oriented design and algorithms

Structure. The backend is split by job: `api/` (HTTP only), `services/` (use cases), `repositories/` (storage), `risk_engine/` (screening, no I/O), `engine/` (pair scoring) and `ingestion/` (data in). Each API module keeps its SQL in a store class injected by FastAPI, so routes are tested with fakes.

- **Abstraction and polymorphism.** Seven data connectors implement one abstract `Connector` class, and the loader treats them alike. Two risk models share one `RiskModel` interface, which lets the system swap the graph model for rules.
- **Dependency inversion.** `RiskOrchestrator` depends on interfaces, not on Neo4j or PyTorch, so a baseline can replace the model with no other change.
- **Encapsulation.** `BehaviourModel` hides calibration and abstention behind two methods; results are frozen dataclasses that cannot change once built.

Algorithm	Where and why	Cost
Breadth-first search	Shortest path to a listed party	O(V + E)
Union-find	Firm groups, so no group crosses the holdout; contact chains	Near-linear
Sliding time window per buyer	Candidate pairs: same buyer within 30 days	O(A log A)
Fellegi-Sunter fusion	Combine evidence so each part stays explainable	O(1) per pair
Gradient-boosted trees, isotonic calibration	Behaviour score with honest probabilities	O(n log n)
SHA-256 hash chain	Tamper-evident audit log	O(1) add, O(n) check

Limits. The engine's main run file is procedural, and some experiment files are long, kept unchanged so every result can be rerun exactly. Suppliers are matched by normalised name; two different firms with the same name would be treated as one. During ingestion, one real supplier was matched by name to a demo record. When the demo records were deleted, its company record went with them. I restored it, and ingestion now ignores demo records. In Kenya, behaviour can only recover links that were disclosed; 75 of the top 100 pairs are not linked, so every flag is a reason to check, not a finding.