Contents

Finnovate – Concept Note

> This document is pre-filled to match the current repository architecture and roadmap. Replace placeholders (like you@example.com) with your details before submission.

Email*

you@example.com

Team Name*

Team Finnovate

Team Leader Name*

Problem Statement*

Small and medium businesses (and early-stage teams) have financial data scattered across exports (CSV/Excel), receipts/PDFs, and app databases. Today, they spend hours reconciling expenses, chasing anomalies, and answering basic finance questions. Tools are either too rigid (pure SQL tables) or too unstructured (files) to support rapid analytics, governance, and explainability together. The result is delayed insights, poor cash discipline, and compliance risks.

Proposed Solution*

Finnovate is a local-first, AI-assisted finance analytics platform that unifies structured, semi-structured, and unstructured data into a single, queryable workspace—backed by a tri-store architecture:

- PostgreSQL (structured): transactions, expenses, vendors, chart-of-accounts, audit logs for integrity and joins.
- MongoDB (semi-structured): flexible documents like raw imports, vendor metadata, enrichment payloads, app events.
- File system (unstructured + vectors): CSV/Parquet drops, PDFs, and vector indexes (ChromaDB/FAISS) for RAG.

Key capabilities (Day 0 → Day N):

- Ingestion: drag/drop CSVs, PDFs, and JSON; automatic schema inference and normalization.
- Validation: Great Expectations checks to catch duplicates, outliers, missing fields before storage.
- Analytics: Streamlit dashboards for burn, runway, vendor spend, cash-in/out, month-over-month deltas.
- Anomaly & trends: scikit-learn based detectors (z-score, isolation forest) and simple time-window deltas.
- RAG assistant: vector search over policies, invoices, statements—"What changed in AWS bill last month?" with cited snippets.
- Observability: MLflow for experiment tracking on models/thresholds; lightweight data lineage via table-level logs.
- DevEx: single bootstrap script, local services by default, Docker used later for final integration and handoff.

Why this is different (and robust):

- The right store for the right data (tri-store) → scalable, cost-effective, and explainable.
- Strict validation before persistence → trustworthy analytics from day one.
- Searchable context with RAG → faster answers, grounded in your own data + documents.
- Pragmatic, incremental ML → start with interpretable, auditable methods; evolve as data grows.

Users*

- Founders/Finance Ops at MSMEs and startups
- Accountants and internal auditors supporting these teams
- Business analysts needing self-serve insights without complex BI setup

Target Audience for the solution provided

- Micro, Small, and Medium Enterprises (MSMEs)
- Early-stage startups and venture-backed teams
- FinOps teams inside product companies
- Fintech partners building on top of our APIs/modules

Expected Outcome*

In a one-week pilot (aligned to the provided Problem Statement and our 6-Day plan), the prototype will deliver the following measurable outcomes. Each outcome maps to the required capabilities and our storage architecture.

1) Data Ingestion & Consolidation (Day 1–2)

- Upload CSV/Excel trial balance and responsibility matrix via Streamlit; parse with Pandas; normalize; persist to PostgreSQL (`users`, `gl_accounts`, `responsibility_matrix`).
- Store raw files under `data/raw/`; write Parquet caches to `data/processed/` per the Storage Architecture.
- Log upload metadata and events in MongoDB (`upload_tracking`/`audit_trail`).
- Success criteria: 90% row-level ingestion success on sample datasets; schema mismatches quarantined with human-readable reasons; ingestion of 10–20k rows completes in 2 minutes on a typical laptop.

2) Validation & Compliance Checks (Day 2)

- Great Expectations (GX) suite enforces: Debits = Credits (Nil total), required columns present, unique keys for (`account_code`,`entity`,`period`), and duplicate row detection.
- Outcomes materialized into: summary status in PostgreSQL (`review_log`), full JSON results in MongoDB (`validation_results`), and checkpoint artifacts in `data/processed/`.
- Success criteria: Trial balance Nil check passes on clean sample; failing rows are quarantined with reasons; overall validation pass rate 95% on curated samples.

3) Automated Report Generation (Day 2–3)

- Reports produced: pending supporting uploads, reviewed vs pending by stakeholder, SLA deviation vs assigned timelines, GL hygiene, and major variances vs previous period/quarter.
- Computations primarily via SQL in PostgreSQL with Parquet caching for hot queries; charts via Plotly.
- Success criteria: first-time computation 10s on 10–20k rows; cached refresh 5s; CSV export of summary tables enabled.

4) Interactive Visualization & Conversational Interface (Day 3–4)

- Streamlit dashboard with drill-down: entity → GL account → review history; status badges for Pass/Fail/Quarantined.
- Conversational interface (LangChain) to answer queries like "Show variances for Adani Ports for 2025-01" and "Which GLs missed SLA this month?".
- RAG over accounting FAQ, policies, and project docs with ChromaDB vectors; answers include citations and snippets.

- Success criteria: dashboard TTFB 5s (cache hit), interactivity p95 latency 250ms; RAG citation coverage 80% of answers; helpfulness rating 70% thumbs-up in demo runs.

5) Proactive, Agentic Insights (Day 4)

- Automated insight cards generated on page load: e.g., "Expense for current period is 1.6× last month for GL 5100 (AWS), driven by data transfer fees."
- Deterministic tool routing for common questions; safe parameter guards and retries.
- Success criteria: at least 5 coherent, accurate insight cards on the sample dataset; zero silent failures (all errors surfaced with friendly messages).

6) Continuous Learning Loop (Day 3–4)

- When a user corrects a misclassification or mapping in the UI, the model retrains (scikit-learn baseline) and logs to MLflow.
- Success criteria: post-correction accuracy improves by 10% on a held-out validation split; rollback available if regression detected.

7) Auditability & Governance (Day 2–5)

- Every key action (upload, validate, review, approve) logged to MongoDB `audit_trail`, with roll-up summaries in PostgreSQL.
- Reproducibility: same inputs → same outputs; versioned expectation suites; clear reviewer notes in `review_log`.
- Success criteria: 100% of critical actions logged with timestamp, actor, and details; data lineage documented at table level.

8) Local-first Developer Experience; Docker for final test (Ongoing; Day 5–6 for Docker)

- Onboarding on a clean Windows laptop in 60 minutes; all flows operate without Docker.
- Optional Docker Compose used in the final 2 days for integration parity.
- Success criteria: new dev can run end-to-end demo (ingest → validate → report → chat → learn) by following README and bootstrap script.

Key KPIs we will track

- Ingestion success rate (90%); number and percentage of quarantined rows with reasons.
- Trial balance Nil check pass rate; count of violations per period/entity.
- Dashboard performance: time-to-first-dashboard (TTFD) and p95 interactivity latency (targets: 5s hit, 10s miss; p95 250ms).
- Variance/anomaly quality: precision@5 0.6 on labeled samples; top drivers surfaced.
- RAG answer quality: citation coverage 80%; helpfulness 70% thumbs-up; average response time 2s on cached vectors.

- Learning loop benefit: 10% relative improvement in target classification metric post-correction; MLflow run metadata captured (params, metrics, artifacts).
- Audit trail completeness: 100% of key events present; mean time-to-diagnose 2 minutes using logs.

Impact*

- Better cash discipline and reduced leakage for MSMEs with minimal setup.
- Faster insight cycles for founders and finance ops → decisions in hours, not weeks.
- Audit-readiness from day zero: validated data, reproducible transformations, and documented lineage.
- Extensible platform: APIs and modular storage enable new verticals (e.g., procurement analytics, credit risk features).

Any other attachments you would like to add as part of the concept note?

- System Architecture: `docs/Architecture.md` (and the SVG diagram referenced within)
- Storage Design: `docs/Storage-Architecture.md`
- Data & App Modules: see `src/` for `db/`, `data_ingestion.py`, `data_validation.py`, `analytics.py`, and `vector_store.py`
- Environment & Setup: `environment.yml`, `scripts/bootstrap.ps1`, `.env.example`

Appendix: Technical Readiness (for reviewers)

- Local-first: runs on Windows with Python 3.11; Docker reserved for final 2-day integration tests.
- Datastores: PostgreSQL 16, MongoDB 7.0, file-based vectors (ChromaDB/FAISS).
- Tooling: Streamlit UI, pandas, scikit-learn, Great Expectations, MLflow, LangChain.
- Security posture for demo: least privileges in Postgres for app user; secrets via `.env`.
- Roadmap next: deeper forecasting (as data accrues), role-based access, and data contracts across teams.