

SHRUTI: Semantic Hybrid Retrieval for Unified Trustworthy Industrial Intelligence

A unified asset and operations brain for industrial plants. One place for every SOP, OEM manual, safety sheet, and inspection report, cross checked automatically, searchable in plain language, and connected to how a plant actually runs day to day: incident reporting, maintenance timelines, compliance tracking, and manager escalation.

Built for the AI for Industrial Knowledge Intelligence hackathon problem statement.

Try it yourself

There is no need to sign up with an invite code just to poke around. A set of test accounts is already seeded across every role in the system, so you can log in as whichever level you want to check out, from a field technician all the way up to the C-suite. The password for every account below is `12345`.

Email	Password	Plant role	HR role
priya.csuite@example.com	12345	Admin	C-suite
rahul.srmanager@example.com	12345	Engineer	Sr. Manager
meera.srhr@example.com	12345	Viewer	Sr. HR
ananya.jrmanager@example.com	12345	Engineer	Jr. Manager
karthik.jrhr@example.com	12345	Viewer	Jr. HR
divya.internhr@example.com	12345	Viewer	Intern HR
vikram.employee@example.com	12345	Technician	Employee
sanjay.employee@example.com	12345	Engineer	Employee

A couple of things worth knowing before you dive in:

- Plant role controls what data you can see and edit (assets, documents, incidents, the AI copilot). HR role controls what you can see on the org chart and who you're allowed to onboard, so pick an account based on what you actually want to test.
- Log in as `priya.csuite@example.com` if you want the full picture: admin access plus visibility into the entire org chart.
- Log in as `vikram.employee@example.com` if you want to see the system the way most day to day plant staff actually experience it.
- These accounts reset periodically, so treat anything you add through them as temporary.

What the name actually means

Each word here maps to something real in the system, not just a fit to the acronym.

- **Semantic:** the AI copilot and incident similarity search run on vector embeddings, so they match by meaning instead of exact keywords.
- **Hybrid Retrieval:** the system runs two retrieval strategies on purpose, not one for the sake of simplicity. Full text search handles exact term lookups for enterprise search, and vector search handles semantic question answering and similarity matching. Each is used where it's the right tool for the job.
- **Unified:** one system covering document intelligence, maintenance, compliance, and organizational access, instead of four separate tools that don't talk to each other.
- **Trustworthy:** the one engineering choice this project treats as non negotiable. The copilot only answers from retrieved, cited sources, reports a confidence score, and refuses to answer when the documents genuinely don't cover the question, rather than inventing something plausible sounding.
- **Industrial Intelligence:** every feature is built around real industrial workflows (SOPs, OEM manuals, safety manuals, maintenance timelines, regulatory compliance), not a generic document chat template pointed at a different domain.

The problem

Industrial plants run on paper trails. The same pump, reactor or boiler gets described across an SOP, an OEM manual, a safety manual and an inspection report, written at different times by different people, usually without anyone cross checking them against each other. When one document gets updated and the others don't, you end up with a silent contradiction: an SOP says lubricate every 30 days, the OEM manual says 60. An operating procedure caps pressure at 8 bar, the safety manual allows 10. Nobody notices until something breaks.

SHRUTI exists to catch that gap automatically, and to give field teams one place to ask "what does the documentation actually say about this equipment" and get a real, cited answer instead of a guess.

Architecture

Three tiers, each deployed independently, each doing one job.

The database API layer is the only one that ever holds real database credentials. Nothing else touches the database directly, except the UI's own dedicated read only path for simple lookups. The AI layer holds no state of its own and can be redeployed, scaled, or swapped out (already proven working on both Render and Hugging Face Spaces) without touching any data. The UI layer never gets a write path into plant data outside its own login/session tables.

Request flow: reporting an incident that turns out to be a known contradiction

Why this addresses each judging criterion

Innovation

Cross document contradiction detection is the core differentiator here, not something bolted on afterward. Most industrial "AI knowledge base" projects stop at search and chat. SHRUTI actively compares structured parameters extracted from every document type against each other for the same asset, flags how serious the disagreement is (critical versus moderate, based on safety relevance), and keeps a resolvable record of it instead of just showing a live check that disappears the moment you refresh the page.

The RAG copilot is built against hallucination, not just for fluency. Responses are retrieval only, every answer needs a source citation, there's a confidence score derived from how well the retrieved content actually matches the question, a screen against prompt injection in the input, and an honest "not enough evidence" response when the documents don't actually cover what's being asked, instead of the model filling the gap with something that sounds right. That matters a lot more here than in a typical chatbot, because the domain is safety critical.

The QR code entry point matters too. Scanning an asset's tag takes a technician straight to its manuals, spare parts, timeline, safety checklist, and any open contradictions, with nothing to search for. That mirrors how equipment actually gets found on a plant floor.

There's also a real onboarding chain behind the scenes, not just a role flag on a user record. Invite codes are single use, expire in 12 hours, and can only be generated by whoever actually has the authority to bring on someone at that level. They get emailed as auto filled links, and the whole structure feeds a live org chart with activity visibility that respects who reports to whom.

Business Impact

This targets a root cause pattern that shows up again and again in industrial incidents: decisions made against documentation that's outdated or contradicts itself. The system doesn't just store documents, it actively points out the specific disagreement (which two documents, which value, which parameter) so someone can fix it before it causes a failure rather than after.

Compliance checking runs against real regulatory style frameworks and is data driven rather than hardcoded. New regulations get added as rows, not code changes, and every asset's inspection readiness is visible at a glance instead of something someone has to reconstruct by hand before an audit.

Incident escalation actually closes the loop. Reporting an incident doesn't just create a row in a table, it notifies the reporter's manager automatically and escalates further up the chain for anything major or critical, so a real person actually gets sent to deal with it instead of the report sitting unread.

Put simply, it reduces a specific and expensive failure mode: the right information existed somewhere in the building, but nobody managed to cross reference it in time.

Technical Excellence

The three tiers are genuinely separated, each one independently deployable and independently testable. This was verified during development against a real database, a real backend server, and a real API layer running together at the same time, not just unit tested in isolation and hoped for.

The async design is deliberately careful about correctness. CPU heavy work like embedding generation, OCR, and PDF parsing, along with blocking network calls to the vector database, are explicitly pushed off to a thread pool so a single document upload can't freeze request handling for every other user on the system. That's a mistake that shows up often enough in async Python backends that it's worth calling out as something this project specifically guards against.

The RAG setup is built the technically correct way for the language model provider being used. Groq serves inference, not embeddings, so vector generation runs on a separate local embedding model rather than incorrectly assuming one API handles both.

Errors are handled defensively and with actual information. Configuration problems, like a missing environment variable or a mismatched vector index dimension, show up as a specific, fixable message instead of a raw stack trace. That's the difference between a production issue that gets fixed in two minutes and one that turns into a long debugging session.

Data access is concurrency aware too. Independent lookups run in parallel instead of one after another, with limits at the connection pool level protecting the database from being overwhelmed rather than letting requests fan out without any bound.

Scalability

The AI layer holds no state, and that's been proven in practice, not just claimed. It's been deployed to more than one hosting target with no code changes, only configuration, so scaling out is a matter of adding instances rather than rearchitecting anything.

Vector search scales independently of the application itself, since it runs through a managed external vector database rather than an index living inside the application container.

There's a clear, documented line drawn around the read path optimization. Pages that are pure retrieval, like asset lists, document lists, and timelines, read directly from the database and skip two network hops. Anything that involves real logic, like compliance evaluation, contradiction detection, or safety scoring, stays behind the one authoritative implementation in the AI layer. That way a performance shortcut never turns into two different versions of the same logic quietly drifting apart.

The database API layer's design, a single entry point with simple routing, was a deliberate simplification that avoids an entire category of routing bugs when moving between hosting environments. This wasn't theoretical: an earlier path based router broke during a real redeploy, and the simpler routing approach used now would not have had that problem.

User Experience

The interface adapts to who's using it. What a field technician sees and can do is different from what a manager sees, which is different again from what an administrator sees, rather than one dashboard trying to serve everyone at once.

The QR code scan is a genuine bridge between the physical equipment and its digital record, with one action to go from standing in front of a pump to seeing everything known about it.

Plain language question answering means nobody needs to know which document to search or what it's called. You can ask "what's the lubrication interval for this pump" instead of digging through PDFs by hand.

Onboarding is guided rather than something that generates a support ticket. A new hire's first action in the system is a short form with an invite link already filled in.

Failure states are honest throughout. The AI copilot says it doesn't have enough information instead of guessing, and cold start delays on the free tier AI deployment show up as an explicit "this may be waking up" message rather than a plain error with no explanation.

Features

Document Intelligence

- Ingests PDFs, scanned PDFs through OCR, and images, with automatic classification
- Extracts parameters, regulatory references, and locations automatically
- Detects contradictions across documents, scores their severity, and keeps a resolvable record
- Builds a knowledge graph linking assets, documents, people, regulations, and incidents
- Answers questions with a cited, confidence scored copilot that admits when it doesn't know
- Full text search across every asset, document, and incident

Maintenance & Safety

- Automatically generated maintenance timelines: install, inspect, leak, repair, failure, replace
- Root cause analysis that connects failure events to corrective action and OEM guidance

- Safety hazard checks (permits, lockout/tagout, chemical hazards, prior incidents) before work begins
- Semantic incident similarity search that finds past incidents even when the wording is different
- Pattern detection for recurring issues across multiple assets, not just one at a time
- Automatic manager notification when an incident is reported, escalating further for anything serious

Compliance & Audit

- Regulatory compliance checks against a data driven, extensible rule set
- Compliance gap tracking with inspection readiness visible per asset
- A full audit trail of who did what, and when, across the whole system

Access & Oversight

- QR code scanning that takes you straight to an asset's full profile
- AI generated insights per asset and a plant wide dashboard
- A role based organizational hierarchy, from employee up through HR and management levels to leadership, with time bound single use onboarding codes and activity visibility that respects reporting lines

Tech stack

Layer	Technology
UI	PHP 8, vanilla JS, session based auth
AI / API	FastAPI (Python 3.11), async throughout
Embeddings	sentence-transformers (all-MiniLM-L6-v2), run locally
LLM generation	Groq (Llama 3.3 70B)
Vector database	Pinecone (serverless)
Relational database	MySQL / MariaDB
Deployment	Render or Hugging Face Spaces for the AI layer, shared hosting for the PHP layers

Repository structure

shruti/

```

└─ db-layer/                                Database API, single entry point, simple
routing
|   └─ index.php
|   └─ sql/                                Schema and migrations
└─ fastapi-layer/                          AI and business logic layer
|   └─ app/
|       └─ routers/                        One module per feature area
|       └─ rag_pipeline.py                 Retrieval augmented generation
|       └─ pinecone_store.py
|       └─ main.py
|       └─ requirements.txt
└─ ui-layer/                               Frontend
|   └─ pages/
|   └─ config.php                         Shared connections, read helpers, auth
|   └─ assets/
└─ docs/
    └─ user-guide.pdf

```

Getting started

1. Database layer

```

bash

mysql -u <user> -p <your_core_database> < db-layer/sql/01_core_schema.sql
mysql -u <user> -p <your_users_database> < db-layer/sql/02_users_schema.sql
# Run any additional migration files in db-layer/sql/ in date order

```

2. Database API layer (shared hosting)

Upload `db-layer/` to your host, copy `.env.sample` to `.env`, and fill in your database credentials and a shared API key.

3. AI layer (Render or Hugging Face Spaces)

Deploy `fastapi-layer/` as a Docker service. Required environment variables:

```

API_KEY=                # matches the database API layer's key
DB_API_BASE_URL=        # the URL where your database API is reachable
PINECONE_API_KEY=
GROQ_API_KEY=

```

4. UI layer (shared hosting)

Upload `ui-layer/`, copy `.env.sample` to `.env`, and fill in your database credentials, the AI layer's URL, and the shared API key.

Full setup notes, including the read only database user recommended for the direct read path, are in each layer's `.env.sample`.

Honest current limitations

Listed here on purpose rather than glossed over.

- Spreadsheet and email archive ingestion are stubbed with a clear "not implemented" response. The PDF, image, and OCR path is fully built. These two would need real sample files in their target format to build correctly against.
- Parameter extraction uses pattern matching tuned to common industrial phrasing, not a general purpose language model. It's precise when values are stated directly and can miss values buried in longer sentences, which was found and confirmed during development.
- The free tier AI deployment has a cold start delay after 15 minutes of inactivity. The UI shows this explicitly rather than hiding it behind a plain error. Temporary solution is provided currently but can scale to a scale.