**🧾 Final Comment – C1-S4 Logging Framework Standardization**

**Epic:** Consolidatie & Documentatie  
**Status:** ✅ Done  
**Date:** 2025-10-18

**Summary**

All API and worker processes now emit **uniform JSON logs** via one centralized structlog configuration.  
Each API request carries a request\_id; each worker run carries a run\_id.  
Common fields: ts, level, service, worker, request\_id / run\_id, event, optionally location\_id and action\_type.  
All logs are **PII-arm**, compatible with the ai\_logs schema (TDA-10) and metrics/alerts (TDA-20).

**Scope / Deliverables**

| **Component** | **Deliverable** | **Status** |
| --- | --- | --- |
| **Core Config** | app/core/logging.py – centralized structlog config with processors (timestamp UTC, level, service, PII guard) | ✅ |
| **Context Helpers** | app/core/request\_id.py – ContextVars for request\_id and run\_id + with\_run\_id() manager | ✅ |
| **API Integration** | app/main.py – middleware sets request\_id, emits request\_started / request\_ended, returns X-Request-Id | ✅ |
| **Worker Integration** | All workers import central logger, bind worker="...", wrap run in with\_run\_id(), log worker\_started / worker\_finished | ✅ |
| **Alert Daemon** | alert\_bot.py converted to central logger; heartbeat event added for infinite loop design | ✅ |
| **Backward Compat** | app/models/ai.py adds ClassificationResult = AIClassification alias to restore dev\_ai imports | ✅ |

**Acceptance Criteria (Traceability)**

* ✅ All processes log JSON with request\_id (API) or run\_id (workers)
* ✅ Uniform field names (ts, level, service, worker, event, ids)
* ✅ Workers use same logger & bind worker + run\_id
* ✅ Logs are PII-arm and aligned with ai\_logs terminology / TDA-20 metrics

**Evidence / Validation Runs**

**API / Health endpoint**

uvicorn app.main:app --reload

curl -i http://127.0.0.1:8000/health

✅ Response 200 + header X-Request-Id → fd95c6563d4d4d348ee8ff9ccdf6b614

{"method":"GET","path":"/health","service":"api","event":"request\_started",...}

{"status\_code":200,"service":"api","event":"request\_ended",...}

**Worker (Classify)**

python -m app.workers.classify\_bot --limit 5 --dry-run

✅ Logs show:

{"service":"worker","worker":"classify\_bot","run\_id":"26934ca0a8e745d598239294b3760cfd","event":"worker\_started"}

{"event":"ai\_validation\_ok","kind":"classification"}

{"event":"worker\_finished","duration\_ms":9611}

**Worker (Alert Daemon)**

python -m app.workers.alert\_bot

✅ Emits worker\_started, alert\_worker\_config, periodic metrics\_snapshot and worker\_heartbeat (daemon design).

**Definition of Done**

* One central logging config used by API and bots — **Done**
* Documentation + runbook included — **Done**
* End-to-end check (API + worker + alert) produce valid JSON for TDA-20 alerts/KPIs — **Done**

**Next Steps**

1. Add action\_type bindings (e.g. DISCOVERY, VERIFY, CLASSIFY) for granular dashboards.
2. Extend error logs with error\_type / error\_code fields for alert routing.
3. Optional: integrate Uvicorn access logs into same structlog stream.

**References**

* 📘 *The New Testament II* – Observability principles
* 📙 *Backlog* – C1-S4 requirements
* 📗 *TDA-3* – initial API logging setup
* 📗 *TDA-10* – ai\_logs audit trail
* 📗 *TDA-20* – metrics & alerts depending on uniform JSON

✅ **Result:** Logging framework standardization complete and fully operational across API and all workers.