SPRINT PLAN — Evidence Ingestion, Traceability & Attestation Workflows

> **Created:** 2026-02-06 18:25 MST **Baseline commit:** `32e502e` (157/157 tests passing) **Governing doc:** `docs/architecture/REGULATORY_EXECUTION_PLATFORM_ARCHITECTURE.md` **Priority order:** Regulatory Twin → Traceability → Gap Detection → Dashboard

---

CURRENT STATE (as of commit 32e502e)

What EXISTS in the database

Table	Status	Notes
organizations	✅ Created	Base schema
users	✅ Created	Minimal — needs role/profile expansion
products	✅ Created	Has org FK
device_versions	✅ Created	Has regulatory_twin_json JSONB
org_members	✅ Created	Multi-org membership
ai_runs	✅ Created + RLS	Full provenance schema
trace_links	✅ Created + RLS	Polymorphic link table
artifacts	✅ Created + RLS	Evidence object registry
artifact_links	✅ Created + RLS	Evidence-to-target links
attestations	✅ Created + RLS	Human sign-off records

What DOES NOT EXIST yet

Entity (from architecture)	DB Table	Python Model	API	Tests
Intended use	❌	❌	❌	❌
Claims	❌	❌	❌	❌
Hazards	❌	❌	❌	❌

Entity (from architecture)	DB Table	Python Model	API	Tests
Harms	❌	❌	❌	❌
Risk controls	❌	❌	❌	❌
Verification tests	❌	❌	❌	❌
Validation tests	❌	❌	❌	❌
Evidence items	❌	❌	❌	❌
Labeling assets	❌	❌	❌	❌
Submission targets	❌	❌	❌	❌

What EXISTS in Python (but not connected to DB)

Component	Status
DeviceInfo Pydantic model	✅ In-memory only, no persistence
ClassificationEngine	✅ Works, no DB storage
PathwayEngine	✅ Works, no DB storage
ai_runs_logger.py	✅ Writes to ai_runs table
supabase_client.py	✅ Connection helper

SPRINT STRUCTURE

Three sprints, each ~1 week. Each sprint delivers:

- SQL migration(s)
- Python models + persistence layer
- Tests (unit + integration)
- Snapshot update

Engineering principles (from governing architecture):

- Structure first, AI second

- Additive changes only
- Version everything
- No regulatory hallucinations
- Human-in-the-loop mandatory

SPRINT 1 — REGULATORY TWIN CORE ENTITIES

Goal: Create the structured data foundation that makes everything else possible. Every device becomes structured data — not just a Pydantic model in memory.

Deliverable 1A: Core Domain Tables Migration

File: `scripts/migrations/2026-02-07_regulatory_twin_core.sql`

New tables (all org-scoped, all with RLS):

```
intended_uses
  - id (uuid PK)
  - organization_id (FK → organizations)
  - device_version_id (FK → device_versions)
  - statement (text NOT NULL)           -- the intended use statement
  - indications (jsonb DEFAULT '[]')    -- list of indications
  - contraindications (jsonb DEFAULT '[]')
  - target_population (text)
  - use_environment (text)              -- clinical, home, point-of-care
  - created_by (FK → users)
  - version (integer NOT NULL DEFAULT 1)
  - supersedes_id (FK → intended_uses, self-ref for versioning)
  - created_at (timestamptz)

claims
  - id (uuid PK)
  - organization_id (FK → organizations)
  - device_version_id (FK → device_versions)
  - claim_type (text NOT NULL)          -- safety, performance, usability
  - statement (text NOT NULL)
  - evidence_basis (text)               -- clinical, bench, lit review
  - status (text DEFAULT 'draft')       -- draft, under_review, accepted, rejected
  - created_by (FK → users)
  - version (integer NOT NULL DEFAULT 1)
  - supersedes_id (FK → claims)
  - created_at (timestamptz)

hazards
  - id (uuid PK)
  - organization_id (FK → organizations)
  - device_version_id (FK → device_versions)
  - hazard_category (text NOT NULL)     -- electrical, biological, software, use-error
  - description (text NOT NULL)
  - foreseeable_sequence (text)         -- how hazard leads to harm
  - severity (text)                     -- negligible, marginal, critical, catastrophic
  - probability (text)                  -- improbable, remote, occasional, probable, frequent
  - risk_level_pre (text)               -- pre-mitigation: low, medium, high, unacceptable
  - created_by (FK → users)
  - version (integer NOT NULL DEFAULT 1)
  - supersedes_id (FK → hazards)
  - created_at (timestamptz)

harms
```

```
  - id (uuid PK)
  - organization_id (FK → organizations)
  - hazard_id (FK → hazards)
  - harm_type (text NOT NULL)          -- injury, death, misdiagnosis, delay
  - description (text NOT NULL)
  - severity (text NOT NULL)
  - affected_population (text)
  - created_by (FK → users)
  - created_at (timestamptz)

risk_controls
  - id (uuid PK)
  - organization_id (FK → organizations)
  - hazard_id (FK → hazards)
  - control_type (text NOT NULL)       -- design, protective, information
  - description (text NOT NULL)
  - risk_level_post (text)             -- post-mitigation residual risk
  - implementation_status (text DEFAULT 'planned')  -- planned, implemented, verified
  - created_by (FK → users)
  - version (integer NOT NULL DEFAULT 1)
  - supersedes_id (FK → risk_controls)
  - created_at (timestamptz)

verification_tests
  - id (uuid PK)
  - organization_id (FK → organizations)
  - device_version_id (FK → device_versions)
  - test_type (text NOT NULL)          -- bench, software, biocompat, electrical
  - title (text NOT NULL)
  - protocol_ref (text)                -- reference to test protocol doc
  - acceptance_criteria (text NOT NULL)
  - result_summary (text)
  - pass_fail (text)                   -- pass, fail, conditional, pending
  - tested_at (timestamptz)
  - created_by (FK → users)
  - version (integer NOT NULL DEFAULT 1)
  - supersedes_id (FK → verification_tests)
  - created_at (timestamptz)

validation_tests
  - id (uuid PK)
  - organization_id (FK → organizations)
  - device_version_id (FK → device_versions)
  - test_type (text NOT NULL)          -- usability, clinical, simulated_use
  - title (text NOT NULL)
  - protocol_ref (text)
  - acceptance_criteria (text NOT NULL)
  - result_summary (text)
  - pass_fail (text)
  - participant_count (integer)
  - tested_at (timestamptz)
  - created_by (FK → users)
  - version (integer NOT NULL DEFAULT 1)
  - supersedes_id (FK → validation_tests)
  - created_at (timestamptz)

evidence_items
  - id (uuid PK)
  - organization_id (FK → organizations)
  - device_version_id (FK → device_versions)
  - evidence_type (text NOT NULL)     -- test_report, lit_review, clinical_data, standard_ref
  - title (text NOT NULL)
  - description (text)
  - source_ref (text)                 -- external reference (standard number, paper DOI)
  - artifact_id (FK → artifacts)      -- link to stored file/content
  - strength (text)                   -- strong, moderate, weak, insufficient
  - status (text DEFAULT 'draft')
  - created_by (FK → users)
  - version (integer NOT NULL DEFAULT 1)
  - supersedes_id (FK → evidence_items)
  - created_at (timestamptz)

labeling_assets
  - id (uuid PK)
  - organization_id (FK → organizations)
  - device_version_id (FK → device_versions)
  - asset_type (text NOT NULL)        -- ifu, label, packaging, e-labeling
  - title (text NOT NULL)
  - content_ref (text)                -- storage URI or artifact link
  - language (text DEFAULT 'en')
  - regulatory_market (text)          -- CA, US, EU
  - artifact_id (FK → artifacts)
  - status (text DEFAULT 'draft')
  - created_by (FK → users)
  - version (integer NOT NULL DEFAULT 1)
  - supersedes_id (FK → labeling_assets)
  - created_at (timestamptz)
```

```
submission_targets
  - id (uuid PK)
  - organization_id (FK → organizations)
  - device_version_id (FK → device_versions)
  - regulatory_body (text NOT NULL)     -- health_canada, fda, eu_mdr
  - submission_type (text NOT NULL)     -- mdl, 510k, de_novo, pma, ce_mark
  - target_date (date)
  - status (text DEFAULT 'planning')    -- planning, preparing, submitted, approved, rejected
  - reference_number (text)             -- assigned by regulator
  - created_by (FK → users)
  - created_at (timestamptz)
```

Key design decisions:

- Every table has `organization_id` for RLS consistency
- Every mutable table has `version` + `supersedes_id` for immutable versioning (Law 7)
- All use `created_by` FK to users for audit trail
- Hazard→Harm→Risk_Control chain is explicit (not polymorphic) for safety
- Evidence_items link to artifacts (stored files) via FK
- Status fields use text enums (not Postgres enums) for flexibility

Deliverable 1B: Pydantic Models

File: `src/core/regulatory_twin.py`

Pydantic models mirroring every table above. Each model includes:

- Validators for status/type fields
- `.to_db_dict()` method for persistence
- `@classmethod from_db_row()` for reading
- Proper Optional fields for nullable columns

Deliverable 1C: Persistence Layer

File: `src/persistence/twin_repository.py`

CRUD operations for each entity:

- `create_intended_use()`, `get_intended_uses_for_device()`, etc.
- All operations scoped to organization
- Uses existing `supabase_client.py` pattern
- Best-effort (never crashes app on DB failure, matching ai_runs_logger pattern)

Deliverable 1D: Tests

Files:

- `tests/unit/test_regulatory_twin_models.py` — Pydantic validation (40+ tests)
- `tests/unit/test_regulatory_twin_migration.py` — SQL file checks (30+ tests, same pattern as RLS tests)
- `tests/integration/test_twin_persistence.py` — DB round-trip tests

Sprint 1 Exit Criteria

- [ ] Migration creates all 10 new tables
- [ ] RLS enabled on all 10 new tables
- [ ] All tables have version + supersedes_id columns
- [ ] Pydantic models for all 10 entities
- [ ] Persistence CRUD for at least: intended_uses, claims, hazards, risk_controls, evidence_items
- [ ] 200+ total tests passing
- [ ] Snapshot updated
- [ ] Committed to main

SPRINT 2 — TRACEABILITY ENGINE + EVIDENCE INGESTION

Goal: Make trace_links operational. Connect the dots: claim → hazard → risk_control → verification → evidence. This is the PRIMARY WEDGE per governing architecture.

Deliverable 2A: Traceability Service

File: `src/core/traceability.py`

The brain of the system. Implements:

```python

```

```
class TraceabilityEngine:
    """
    Creates and queries regulatory trace links.

    Supports the full chain:
      claim → mitigated_by → risk_control
      risk_control → verified_by → verification_test
      verification_test → supported_by → evidence_item
      hazard → causes → harm
      hazard → mitigated_by → risk_control
      claim → supported_by → evidence_item
    """

    VALID_RELATIONSHIPS = {
        ("claim", "hazard"): ["addresses"],
        ("claim", "evidence_item"): ["supported_by"],
        ("hazard", "harm"): ["causes", "may_cause"],
        ("hazard", "risk_control"): ["mitigated_by"],
        ("risk_control", "verification_test"): ["verified_by"],
        ("risk_control", "validation_test"): ["validated_by"],
        ("verification_test", "evidence_item"): ["supported_by"],
        ("validation_test", "evidence_item"): ["supported_by"],
        ("evidence_item", "artifact"): ["documented_in"],
    }

    def create_link(self, source_type, source_id, target_type, target_id,
                    relationship, rationale, created_by) -> TraceLink

    def get_links_from(self, source_type, source_id) -> List[TraceLink]

    def get_links_to(self, target_type, target_id) -> List[TraceLink]

    def get_full_chain(self, claim_id) -> TraceChain
        """Follow all links from a claim down to evidence."""

    def get_coverage_report(self, device_version_id) -> CoverageReport
        """For each claim, show: linked hazards, controls, tests, evidence."""

    def validate_link(self, source_type, target_type, relationship) -> bool
        """Check if the relationship is valid per VALID_RELATIONSHIPS."""
```

Deliverable 2B: Evidence Ingestion Service

File: `src/core/evidence_ingestion.py`

```python
class EvidenceIngestionService:
    """
    Ingest evidence and connect it to the regulatory twin.

    Workflow:
      1. Create artifact (file metadata + content hash)
      2. Create evidence_item (typed, with strength assessment)
      3. Create trace_link to the relevant claim/test/control
      4. Optionally log AI assist via ai_runs if AI helped classify
    """

    def ingest_evidence(self, device_version_id, evidence_type, title,
                        artifact_data, linked_to) -> EvidenceItem

    def bulk_ingest(self, device_version_id, items: List[dict]) -> List[EvidenceItem]

    def get_evidence_for_claim(self, claim_id) -> List[EvidenceItem]

    def get_unlinked_evidence(self, device_version_id) -> List[EvidenceItem]
        """Find evidence items not connected to any claim/test/control."""
```

Deliverable 2C: Attestation Workflow Service

File: `src/core/attestation_service.py`

```python
```

```
class AttestationService:
    """
    Human-in-the-loop sign-off on artifacts and links.

    Every AI output, every trace link, every evidence assessment
    can be attested to by a human reviewer.
    """

    ATTESTATION_TYPES = [
        "reviewed",        # human reviewed the content
        "approved",        # human approved for regulatory use
        "rejected",        # human rejected, needs rework
        "acknowledged",    # human saw it, no opinion
    ]

    def attest_artifact(self, artifact_id, attested_by, attestation_type, note)

    def attest_link(self, artifact_link_id, attested_by, attestation_type, note)

    def get_attestation_status(self, artifact_id) -> AttestationStatus

    def get_unattested_items(self, organization_id) -> List[dict]
        """Find artifacts and links that have not been reviewed."""

    def get_attestation_audit_trail(self, artifact_id) -> List[Attestation]
```

Deliverable 2D: API Endpoints

File: `src/api/traceability_routes.py`

New endpoints (added to existing FastAPI app):

```
POST   /api/v1/trace-links          — create a trace link
GET    /api/v1/trace-links/{id}     — get a trace link
GET    /api/v1/trace-chains/{claim_id} — get full chain from claim to evidence
GET    /api/v1/coverage/{device_version_id} — coverage report

POST   /api/v1/evidence             — ingest evidence item
GET    /api/v1/evidence/{device_version_id} — list evidence for device
GET    /api/v1/evidence/unlinked/{device_version_id} — unlinked evidence

POST   /api/v1/attestations         — create attestation
GET    /api/v1/attestations/pending/{org_id} — unattested items
GET    /api/v1/attestations/trail/{artifact_id} — audit trail
```

Deliverable 2E: Tests

Files:

- `tests/unit/test_traceability.py` — link validation, chain traversal (40+ tests)
- `tests/unit/test_evidence_ingestion.py` — ingest, bulk, unlinked detection (25+ tests)
- `tests/unit/test_attestation_service.py` — attest, reject, audit trail (25+ tests)
- `tests/integration/test_trace_chain_flow.py` — full claim→evidence flow (10+ tests)
- `tests/api/test_traceability_endpoints.py` — API route tests (20+ tests)

Sprint 2 Exit Criteria

☐ TraceabilityEngine creates/queries links with validation
☐ Full chain traversal: claim → hazard → control → test → evidence
☐ Evidence ingestion with artifact creation
☐ Attestation workflow (attest, reject, audit trail)
☐ API endpoints for all three services
☐ Coverage report per device version
☐ Unlinked evidence detection
☐ 320+ total tests passing
☐ Snapshot updated
☐ Committed to main

SPRINT 3 — GAP DETECTION ENGINE + READINESS ASSESSMENT

Goal: The highest-value feature. Rules that surface what's missing, weak, or inconsistent. Per architecture:

"Never say 'You are submission ready.' Always say 'Readiness assessment based on configured expectations.'"

Deliverable 3A: Gap Detection Rules Engine

File: `src/core/gap_engine.py`

```python
class GapDetectionEngine:
    """
    Rules-based engine that evaluates regulatory readiness.

    Rules are versioned, deterministic, and explainable.
    Each rule produces a GapFinding with severity, description, and remediation.
    """

    def evaluate(self, device_version_id) -> GapReport:
        """Run all rules against a device version."""

    def evaluate_rule(self, rule_id, device_version_id) -> List[GapFinding]:
        """Run a single rule."""

    def get_rules(self) -> List[GapRule]:
        """List all active rules with descriptions."""


class GapRule:
    """A single detection rule. Deterministic. Explainable."""
    id: str
    name: str
    description: str
    severity: str       # critical, major, minor, info
    category: str       # coverage, completeness, consistency, evidence_strength
    version: int
    evaluate: Callable    # returns List[GapFinding]
```

Initial rule set (Health Canada focus):

Rule ID	Name	What it checks	Severity
GAP-001	Unmitigated hazards	Hazards with no linked risk_control	CRITICAL
GAP-002	Unverified controls	Risk controls with no linked verification_test	CRITICAL
GAP-003	Unsupported claims	Claims with no linked evidence_item	MAJOR
GAP-004	Missing intended use	Device version with no intended_use record	CRITICAL
GAP-005	Weak evidence	Evidence items with strength = 'weak' or 'insufficient'	MAJOR
GAP-006	Untested claims	Claims with no linked verification OR validation test	MAJOR
GAP-007	No submission target	Device version with no submission_target	MINOR
GAP-008	Unattested AI outputs	ai_runs linked to artifacts but not attested	MAJOR
GAP-009	Missing labeling	Device version with no labeling_assets	MAJOR
GAP-010	Incomplete risk chain	Hazard → harm → control chain has breaks	CRITICAL
GAP-011	Draft evidence only	All evidence_items in 'draft' status	MAJOR
GAP-012	No clinical evidence (Class III/IV)	Class III/IV with no clinical evidence type	CRITICAL

Deliverable 3B: Readiness Assessment

File: `src/core/readiness.py`

```python
```

```
class ReadinessAssessment:
    """
    Aggregates gap findings into a readiness score.

    NEVER says "compliant" or "ready."
    ALWAYS says "Readiness assessment based on configured expectations."
    """

    def assess(self, device_version_id) -> ReadinessReport:
        """
        Returns:
        - overall_readiness_score: float (0.0 - 1.0)
        - category_scores: dict[str, float]
        - gap_findings: List[GapFinding]
        - critical_blockers: List[GapFinding]
        - summary: str  (regulatory-safe language)
        """

    def generate_summary(self, report: ReadinessReport) -> str:
        """
        Generates human-readable summary.
        Uses ONLY approved regulatory language.
        """
```

Deliverable 3C: API Endpoints

```
GET  /api/v1/gaps/{device_version_id}          — full gap report
GET  /api/v1/gaps/{device_version_id}/critical — critical gaps only
GET  /api/v1/readiness/{device_version_id} — readiness assessment
GET  /api/v1/rules                             — list all gap rules
```

Deliverable 3D: Tests

Files:

- `tests/unit/test_gap_engine.py` — each rule individually tested (50+ tests)
- `tests/unit/test_readiness.py` — scoring, language safety (20+ tests)
- `tests/regulatory/test_gap_rules.py` — regulatory correctness of rules (15+ tests)
- `tests/integration/test_gap_detection_flow.py` — end-to-end (10+ tests)
- `tests/api/test_gap_endpoints.py` — API route tests (10+ tests)

Sprint 3 Exit Criteria

- [ ] 12 gap detection rules implemented and tested
- [ ] Each rule produces explainable findings with severity
- [ ] Readiness assessment with category scores
- [ ] ALL output uses regulatory-safe language (no "compliant", no "ready")
- [ ] API endpoints for gaps, readiness, rules
- [ ] 430+ total tests passing
- [ ] Snapshot updated
- [ ] Committed to main

CUMULATIVE MILESTONE TRACKER

Metric	Baseline (now)	Sprint 1	Sprint 2	Sprint 3
DB tables	9	19	19	19
RLS-enabled tables	9	19	19	19
Python models	6	16	16	18
Service classes	2	3	6	8
API endpoints	6	6	15	19
Total tests	157	230+	320+	430+
Trace link types	0	0	9	9
Gap rules	0	0	0	12

TECH DEBT TO ADDRESS ALONGSIDE

Item	Priority	When
34 mypy errors	MEDIUM	Fix progressively per sprint
`public.users` vs `auth.users` decision	HIGH	Before Sprint 1 migration
Supabase cloud deployment	HIGH	After Sprint 2 (need real auth for RLS testing)

Item	Priority	When
S3 document storage	MEDIUM	Sprint 2 (evidence needs file storage)
Pre-commit mypy enforcement	LOW	After mypy errors resolved

RISK REGISTER

Risk	Impact	Mitigation
Schema changes break existing tests	HIGH	All migrations idempotent, test existing 157 before each sprint
Supabase auth.users vs public.users confusion	HIGH	Decide BEFORE Sprint 1, document in CLAUDE.md
Scope creep on Regulatory Twin entities	MEDIUM	Stick to architecture entities only, no extras
Gap rules too rigid	LOW	Version rules, allow disable/enable per org
No real multi-user testing	HIGH	Supabase cloud deployment before Sprint 3

WHAT THIS PLAN DOES NOT INCLUDE (deferred)

These are Phase 4-5 per architecture, deliberately excluded:

- Document Orchestration (generating submission documents from structured data)
- Deficiency Response Copilot
- Regulatory Knowledge Graph construction
- Real UI/dashboard (Streamlit or React)
- AI-assisted link recommendations
- S3 signed URL document storage

Reason: The architecture says "Earn complexity." We build the data foundation and deterministic logic first. AI and UI come after structure is solid.

End of sprint plan