2) MVP Technical Specification — Architecture, Data Model, APIs, Pipelines, Controls

2.1 System architecture (MVP)

Components

1. **Web App**
    - Admin console + analytics dashboards
2. **API Gateway**
    - AuthN/AuthZ, request logging, tenant scoping
3. **Ingestion Service**
    - uploads, mapping templates, validation, canonicalization
4. **Geospatial Service**
    - geocoding, hazard overlays
5. **Analytics Service**
    - rollups, thresholds, breaches, drift
6. **Governance Service**
    - version registry, run registry, lineage, audit events

Storage

- **PostgreSQL + PostGIS**: canonical data + spatial queries
- **Object storage** (S3-compatible): raw uploads, parquet extracts, run artifacts, exports

- **Queue/orchestrator**: background jobs for geocoding, overlays, rollups

Determinism principle (non-negotiable)

Every computed output must be reproducible from:

- immutable inputs (versions)
- immutable configs
- pinned dataset versions
- recorded code/pipeline version (build hash)

2.2 Core data model (entities and key fields)

Tenant & Users

- `tenant(id, name, created_at)`
- `user(id, tenant_id, email, role, status, created_at)`
- `role` is coarse in MVP: `ADMIN`, `ANALYST`, `OPS`, `AUDITOR`, `READ_ONLY`

Exposure ingestion

- `exposure_upload(id, tenant_id, filename, storage_uri, uploaded_by, uploaded_at, status)`
- `mapping_template(id, tenant_id, name, version, mapping_json, created_by, created_at)`
- `validation_result(id, upload_id, summary_json, row_errors_uri, created_at)`
- `exposure_version(id, tenant_id, name, source_upload_id, created_by, created_at, immutable=true)`

Canonical objects

- `location(id, tenant_id, exposure_version_id, external_location_id, address_fields..., lat, lon, geocode_confidence, quality_tier, quality_reasons_json, tiv, limit, premium, currency, lob, occupancy, construction, year_built, policy_id, account_id, updated_at)`

- `account(id, tenant_id, exposure_version_id, external_account_id, name, attributes_json)`

- `policy(id, tenant_id, exposure_version_id, external_policy_id, inception_date, expiry_date, attributes_json)`

Hazard datasets and overlays

- `hazard_dataset(id, name, peril, vendor, coverage_geo, license_ref)`

- `hazard_dataset_version(id, hazard_dataset_id, version_label, storage_uri, checksum, effective_date, created_at)`

- `hazard_overlay_result(id, tenant_id, exposure_version_id, hazard_dataset_version_id, method, params_json, created_at)`

- `location_hazard_attribute(location_id, hazard_overlay_result_id, attributes_json)`
  (attributes_json contains standardized keys like `band`, `percentile`, `score`, `category`)

Analytics & controls

- `rollup_config(id, tenant_id, name, dimensions_json, filters_json, measures_json, created_by, created_at, version)`

- `rollup_result(id, tenant_id, exposure_version_id, rollup_config_id, hazard_overlay_result_ids_json, storage_uri,`

`created_at)`

- `threshold_rule(id, tenant_id, name, rule_json, severity, created_by, created_at, active)`

- `breach(id, tenant_id, exposure_version_id, threshold_rule_id, rollup_key_json, metric_value, threshold_value, created_at, status)`

Drift

- `drift_run(id, tenant_id, exposure_version_a, exposure_version_b, config_json, storage_uri, created_at)`

Governance

- `run(id, tenant_id, run_type, input_refs_json, config_refs_json, output_refs_json, code_version, status, created_by, created_at)`

- `audit_event(id, tenant_id, actor_user_id, action, entity_type, entity_id, event_json, created_at)`
  (append-only, never updated)

2.3 Exposure Data Contract v1 (minimum viable)

Required fields (per location row)

- `external_location_id` (string; stable within source system)

- Address: one of:

  - `lat`, `lon` (preferred) OR

  - `address_line1`, `city`, `state_region`, `postal_code`, `country`

- Financial:

    - `tiv` (numeric, >= 0)
    - `currency` (ISO code)
- Segmentation (at least one):
    - `lob` or `product_code`

Optional (strongly encouraged)

- `limit`, `premium`
- `occupancy`, `construction`, `year_built`
- `account_id`, `policy_id`
- `inception_date`, `expiry_date`

Validation rules (examples)

- ERROR:
    - Missing `external_location_id`
    - Missing both (lat/lon) and sufficient address fields
    - `tiv` missing or negative
- WARN:
    - Low geocode confidence
    - Currency missing (default tenant currency applied)
    - `year_built` outside plausible range
- INFO:
    - occupancy unknown

    - construction unknown

2.4 Data quality scoring rubric (transparent tiers)

Compute three sub-scores then assign tier.

- **Completeness score (0–100)**
    - Required fields present
    - Optional enrichers present (segmentation fields, policy/account linkage)
- **Geocode score (0–100)**
    - Confidence from geocoder
    - Lat/lon provided vs inferred
- **Financial sanity score (0–100)**
    - Non-negative, non-null
    - Outlier checks (tenant-configured bounds)

Tier assignment

- Tier A: >= 85 overall and geocode >= 80
- Tier B: >= 70 overall and geocode >= 60
- Tier C: otherwise
  Store `quality_reasons[]` as human-readable codes (e.g., `MISSING_POSTAL`, `LOW_GEOCODE_CONFIDENCE`).

2.5 Hazard overlay interface (MVP)

Standardize hazard attributes across datasets:

Minimum attribute keys per peril proxy:

- `hazard_category` (string; e.g., `FLOOD`, `WILDFIRE`)
- `band` (integer or string; e.g., 1–5)
- `percentile` (0–100, optional)
- `score` (numeric, optional)
- `source` (dataset + version)
- `method` (join/sample/lookup)

This enables consistent rollups even when underlying datasets differ.

2.6 API specification (MVP endpoints)

Auth & tenancy

- `POST /auth/login` (SSO can be later; MVP can start with enterprise-friendly auth provider)
- All requests require `tenant_id` scope (from token claims)

Upload and mapping

- `POST /uploads` → returns `upload_id` and signed upload URL (or direct upload)
- `POST /uploads/{upload_id}/mapping` → create/attach mapping template
- `POST /uploads/{upload_id}/validate` → returns validation summary + job id
- `POST /uploads/{upload_id}/commit` → creates `exposure_version_id` (immutable)

Exposure versions

- `GET /exposure-versions`

- `GET /exposure-versions/{id}/summary`
- `GET /exposure-versions/{id}/locations?filters=...` (paged)
- `GET /exposure-versions/{id}/exceptions` (quality Tier C, geocode low confidence, validation WARN/ERROR remnants)

Hazard overlays

- `POST /hazard-overlays` with `{exposure_version_id, hazard_dataset_version_ids[], params}`
- `GET /hazard-overlays/{id}/status`
- `GET /hazard-overlays/{id}/summary`

Rollups

- `POST /rollup-configs`
- `POST /rollups` with `{exposure_version_id, rollup_config_id, hazard_overlay_result_ids[]}`
- `GET /rollups/{id}` (returns metadata + storage URI or direct data)
- `GET /rollups/{id}/drilldown?rollup_key=...` → list locations/accounts contributing

Thresholds and breaches

- `POST /threshold-rules`
- `POST /breaches/run` with `{exposure_version_id, threshold_rule_ids[], rollup_result_id}`
- `GET /breaches?exposure_version_id=...`

- `PATCH /breaches/{id}` (status workflow: `OPEN`, `ACKED`, `RESOLVED`)

Drift

- `POST /drift` with `{exposure_version_a, exposure_version_b, config}`
- `GET /drift/{id}` (summary)
- `GET /drift/{id}/details` (by dimension + lists)

Governance

- `GET /runs/{id}` (inputs/config/outputs + code_version)
- `GET /lineage?entity_type=rollup_result&entity_id=...`
- `GET /audit-events?filters=...`

Example payload: create rollup config

```
{
  "name": "Accumulation by state x wildfire band x LOB",
  "dimensions": ["state_region", "lob", "hazard.WILDFIRE.band",
"quality_tier"],
  "filters": { "currency": "USD" },
  "measures": ["sum_tiv", "sum_limit", "count_locations"]
}
```

2.7 Pipeline design (idempotent, observable)

Pipeline: upload → version

1. Store raw file (object storage) + metadata
2. Apply mapping template (deterministic transform)
3. Validate (produce summary + row-level errors file)

4. Canonicalize rows → write to Postgres tables partitioned by `exposure_version_id`

5. Create immutable `exposure_version`

Idempotency key: `(tenant_id, upload_id, mapping_template_version)`
If rerun, produces same exposure_version or a new version with explicit reason.

Pipeline: geocode + quality scoring

1. Standardize addresses

2. Geocode missing lat/lon

3. Compute sub-scores + tier + reasons

4. Write updates to location records (or write a separate immutable "enrichment result" if you want stricter immutability—either is acceptable in MVP, but keep versioned lineage)

Pipeline: hazard overlays

1. For each dataset version, run spatial join/sample

2. Store results with `hazard_overlay_result_id`

3. Materialize location-level hazard attributes

Pipeline: rollup

1. Read locations for exposure_version + overlays

2. Aggregate according to `rollup_config`

3. Store result (table or parquet in object storage) with metadata row in DB

Pipeline: breaches

1. Evaluate threshold rules against rollup result

2. Create breach records with rollup keys + metric comparisons

Pipeline: drift

1. Diff location sets by `external_location_id` (or canonical hash)
2. Attribute changes: NEW / REMOVED / MODIFIED
3. Aggregate diffs by configured dimensions
4. Compare breaches between versions

2.8 Governance model (runs, lineage, audit)

Run object (required for any computed output)

- `run_type`: `VALIDATION`, `GEOCODE`, `OVERLAY`, `ROLLUP`, `BREACH_EVAL`, `DRIFT`
- `input_refs`: exposure_version_id, hazard_dataset_version_ids, prior versions
- `config_refs`: mapping_template_version, rollup_config_id, threshold_rule_id(s)
- `output_refs`: rollup_result_id, overlay_result_id, breach_ids, drift_run_id
- `code_version`: build hash
- `status`: `QUEUED`, `RUNNING`, `SUCCEEDED`, `FAILED`

Audit events (append-only)

Emit audit events for:

- login, role changes
- mapping template changes
- exposure version commit

- hazard dataset version registration
- threshold creation/activation
- breach status changes
- exports generated

2.9 Security and compliance baseline (MVP-ready)

- Tenant isolation enforced at:
    - JWT claims → API gateway → DB row-level scoping (or schema-per-tenant)
- RBAC:
    - `ADMIN`: manage users/configs
    - `OPS`: upload/mapping/exception workflows
    - `ANALYST`: overlays/rollups/thresholds/drift
    - `AUDITOR`: read-only + lineage/audit visibility
- Encryption:
    - TLS in transit
    - at-rest encryption for object storage and DB
- Secrets management:
    - managed secret store; no secrets in env files
- Logging:
    - structured logs with correlation IDs
    - audit events stored separately from app logs
- Data retention:

    - configurable retention for raw uploads and exports per tenant policy

2.10 Observability and reliability (minimum viable)

- Metrics:
    - pipeline duration by run_type
    - failure rate by step
    - geocode coverage %
    - tier distribution
    - rollup compute time by portfolio size
- Alerts:
    - pipeline failures
    - backlog queue depth
- SLO targets (internal):
    - rollup for typical portfolio size completes within defined target (set per tenant size class)

2.11 Testing strategy (must-have)

- Unit tests:
    - mapping transforms
    - validation rules
    - rollup dimension logic
- Golden dataset tests:
    - fixed sample exposure file → expected rollups/breaches

- Determinism tests:
    - same inputs/config produce identical outputs checksums
- Security tests:
    - tenant boundary tests (cannot query another tenant's objects)

---

3) Build plan as an execution backlog (first 6–8 weeks of work, sequence-based)

Milestone A — Trusted Exposure

- Implement upload + mapping templates + validation engine
- Canonical schema + version registry
- Exceptions report export
- Basic UI: upload, mapping, validation

Milestone B — Geocode + Quality

- Geocoding pipeline + confidence scoring
- Quality rubric + tier assignment + exceptions queue
- UI: exceptions queue + export

Milestone C — Overlays + Rollups

- Hazard dataset registry + dataset versioning
- Overlay pipeline + standardized hazard attributes

- Rollup configs + rollup execution + dashboard
- UI: accumulation dashboard + drill-down

Milestone D — Thresholds + Drift + Governance hardening

- Threshold builder + breach evaluation + breach workflow
- Drift reports and breach deltas
- Lineage and run views + audit viewer
- Exportable drift/breach reports