Intelligent Cloud Platform
CMPE - 281
Project Deliverable 2

Component Design Document

Project: Intelligent Cloud Platform for Smart Homes

Instructor: Dr. Jerry Gao

Team:

Vijaya Sharavan Reddy Baddam - 018321342
Venkata Gowtham Jalam - 018315791
Nikhil Dupally - 018325736
Divyasri Lakshmi Alekhya Nakka - 018291702

Table Of Contents

System Dashboard -- Vijaya Sharavan Reddy Baddam

Section 1: Component overview

Purpose: provide a single, role-aware view of fleet health, alert load, SLA compliance, device connectivity, model status, and tenant usage.
Objectives:

- real-time KPIs (alerts/min, open vs. resolved, ack/resolve SLA)
- fleet health (online/offline devices, firmware distribution)
- model/service status (latency, error rates)
- multi-tenant usage, billing-grade metrics

Function scope: dashboards, drill-downs, saved reports, exports (CSV), access control, audit trail.
Usage: homeowners (personal status), cloud staff (ops KPIs), IoT team (device health).

Section 2: Component APIs (REST)

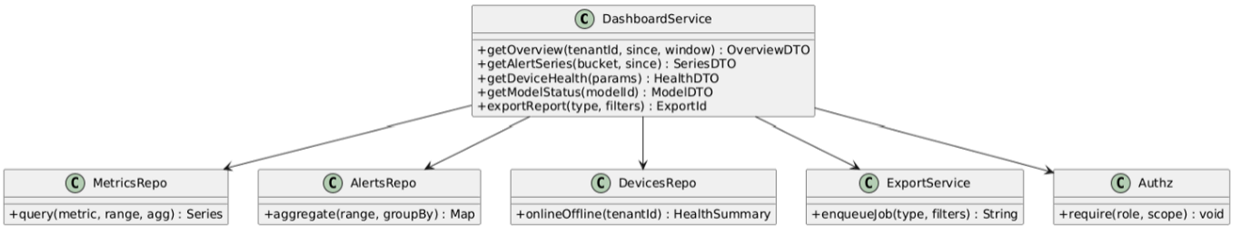
Base path /api/v1/dashboard (JWT, role scoped, 60 rpm/user).

- GET /overview?tenantId=&since=&window= → totals: alerts, open, ack SLA %, resolve SLA %, devices online/offline, model latency p95.
- GET /alerts/timeseries?bucket=1m&since= → [{ts, new, acked, resolved}]
- GET /devices/health?tenantId=&groupBy=house|model|firmware → online/offline %, lastSeen, errorTopN.
- GET /models/status?modelId= → p50/p95/p99 latency, accuracy snapshot (if available), errorRate.
- GET /reports/usage?tenantId=&period=day|week|month → events, API calls, storage GB, egress GB.
- POST /exports body: {type:"alerts|devices|usage", filters:{...}} → 202 + export id.
- GET /exports/{id} → signed download url (CSV).

Section 3: Function & data design, behavior

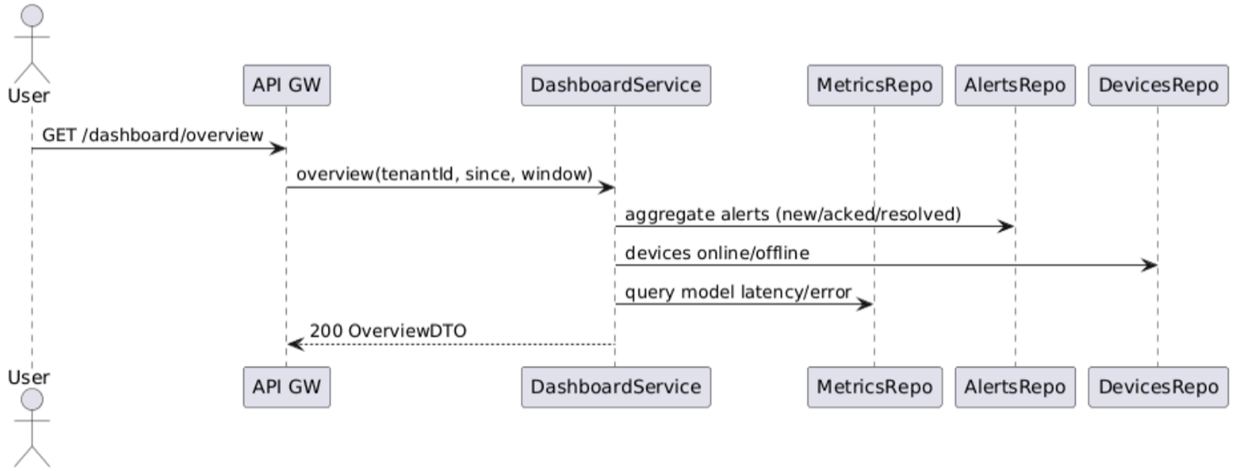
Class diagram

DashboardService caches KPIs, reads alert/device/metrics stores, and exports via S3.



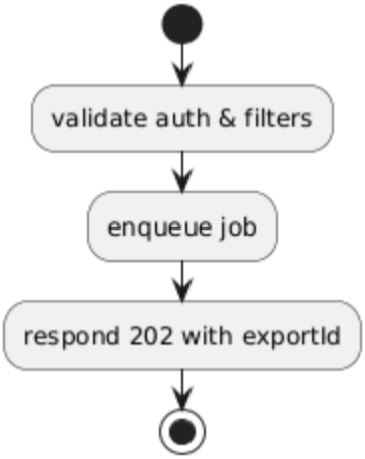
Sequence — overview KPI fetch

GET /overview → cache check → aggregate on miss → return totals and latency.



Flowchart — export

User requests export → build CSV → upload to S3 → return time-boxed signed URL.



Data: read-only from alerts/devices/model metrics stores; exports written to S3 with signed URLs; cache hot KPIs in Redis (TTL ~30–60s).

Section 4: Business logic (decision tables)

Widget visibility (role + data)

Role	Tenant Scope	Show Billing	Show Model internals
Homeowner	own	No	No
Cloud Staff	any	Yes	Yes
IoT Team	any	No	Partially (latency only)

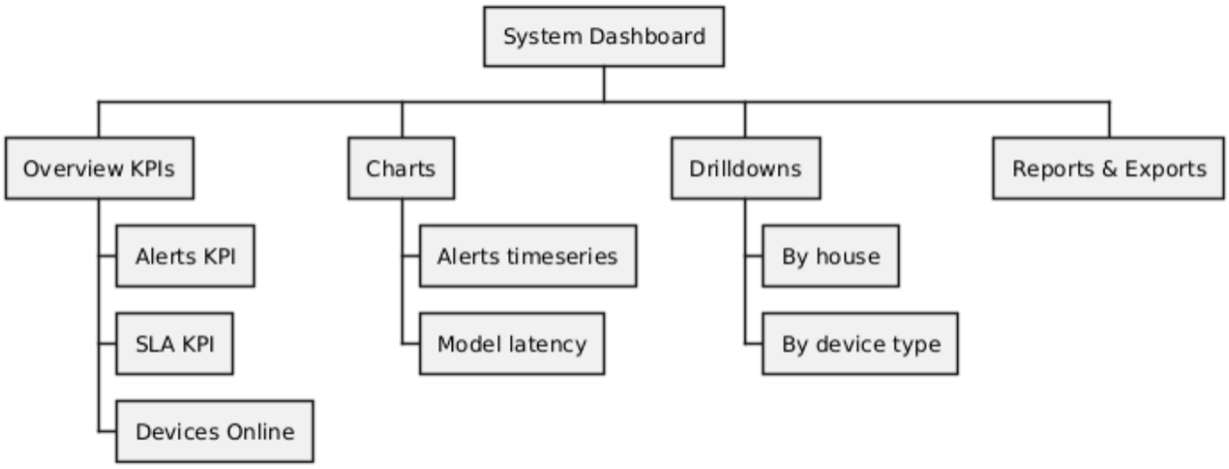
SLA coloring

KPI	Thresholds	Color
Ack SLA	≥95%=Green; 80–95%=Amber; <80%=Red	traffic

Section 5: GUI design

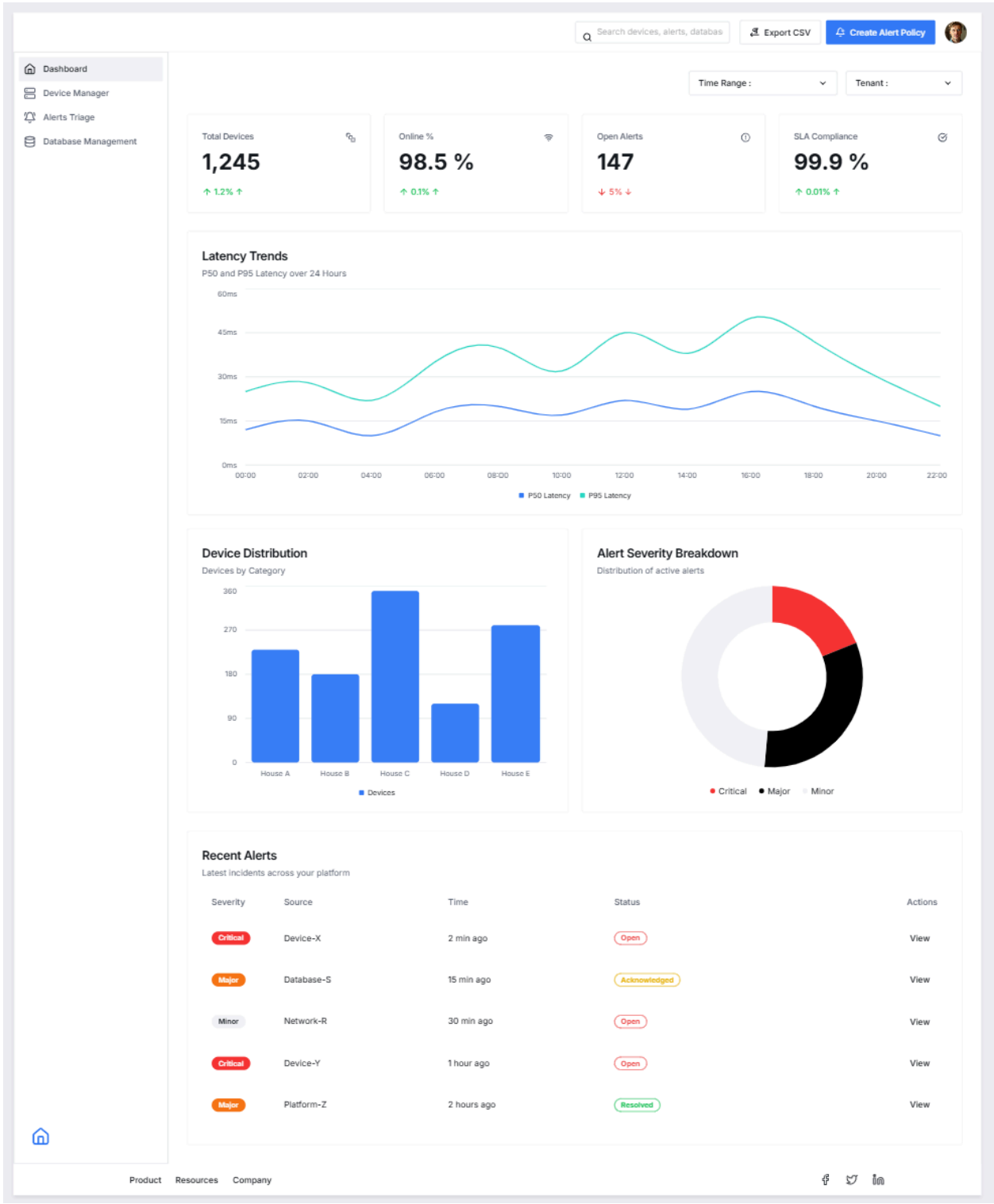
Style: responsive cards; dark/light; badges by severity; loading skeletons; filters on top.

Function partition tree



Storyboard: land on KPIs → pick time range → hover chart → click drill-down to house → open table → export.

UI SALT mockup


Search devices, alerts, databas
Export CSV
Create Alert Policy
Dashboard
Device Manager
Alerts Triage
Database Management
Time Range :
Tenant :
Total Devices
1,245
↑ 1.2% ↑
Online %
98.5 %
↑ 0.1% ↑
Open Alerts
147
↓ 5% ↓
SLA Compliance
99.9 %
↑ 0.01% ↑
Latency Trends
P50 and P95 Latency over 24 Hours
60ms
45ms
30ms
15ms
0ms
00:00
02:00
04:00
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
P50 Latency
P95 Latency
Device Distribution
Devices by Category
360
270
180
90
0
House A
House B
House C
House D
House E
Devices
Alert Severity Breakdown
Distribution of active alerts
Critical
Major
Minor
Recent Alerts
Latest incidents across your platform
Severity
Source
Time
Status
Actions
Critical
Device-X
2 min ago
Open
View
Major
Database-S
15 min ago
Acknowledged
View
Minor
Network-R
30 min ago
Open
View
Critical
Device-Y
1 hour ago
Open
View
Major
Platform-Z
2 hours ago
Resolved
View
Product
Resources
Company

Device Manager -- Venkata Gowtham Jalam

Section 1: Component overview

Purpose: onboard, configure, control, and monitor devices per house/room (cameras, mics, smoke sensors).
Objectives: zero-touch provisioning, Wi-Fi pairing, config profiles, OTA firmware, health monitoring, and remote controls (PTZ, reboot).
Function scope: device registry, pairing, config templates, commands, firmware mgmt, connectivity tests, diagnostics logs.
Usage: IoT team & homeowners (limited).

Section 2: APIs (REST + commands)

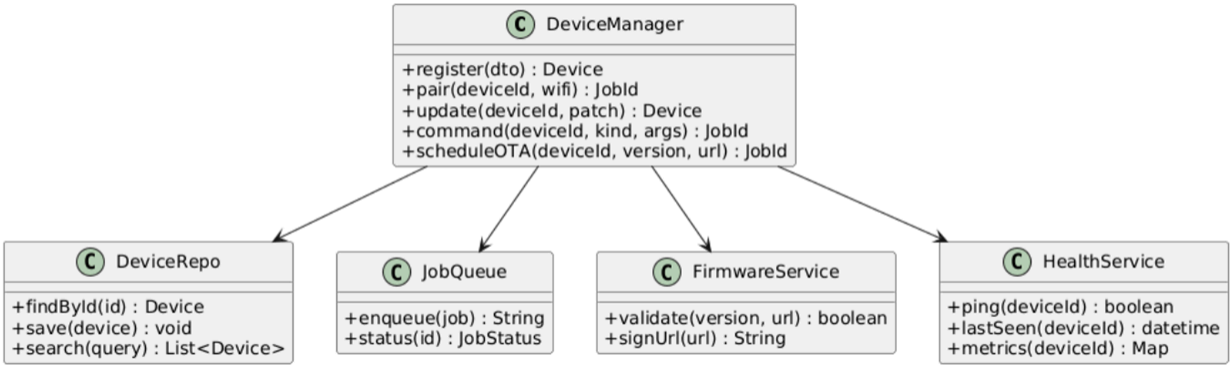
Base /api/v1/devices (JWT + RBAC; device commands are async).

- POST /register {serial, type, model, houseId, room, firmware} → 201 {deviceId, claimCode}
- POST /pair {deviceId, wifiSsid, wifiPwd} → 202 (pairing job)
- GET /{deviceId} → device profile + lastSeen + health
- PATCH /{deviceId} body {room?, name?, config?}
- POST /{deviceId}/command {kind:"REBOOT|PTZ|LED|BEEP", args:{}} → 202
- POST /{deviceId}/firmware {version, url} → 202 (OTA)
- GET /{deviceId}/jobs?since= → command/OTA job statuses
- GET /search?houseId=&type=&state= → list
- POST /bulk/config {deviceIds[], config} → 202

Section 3: Function & data design

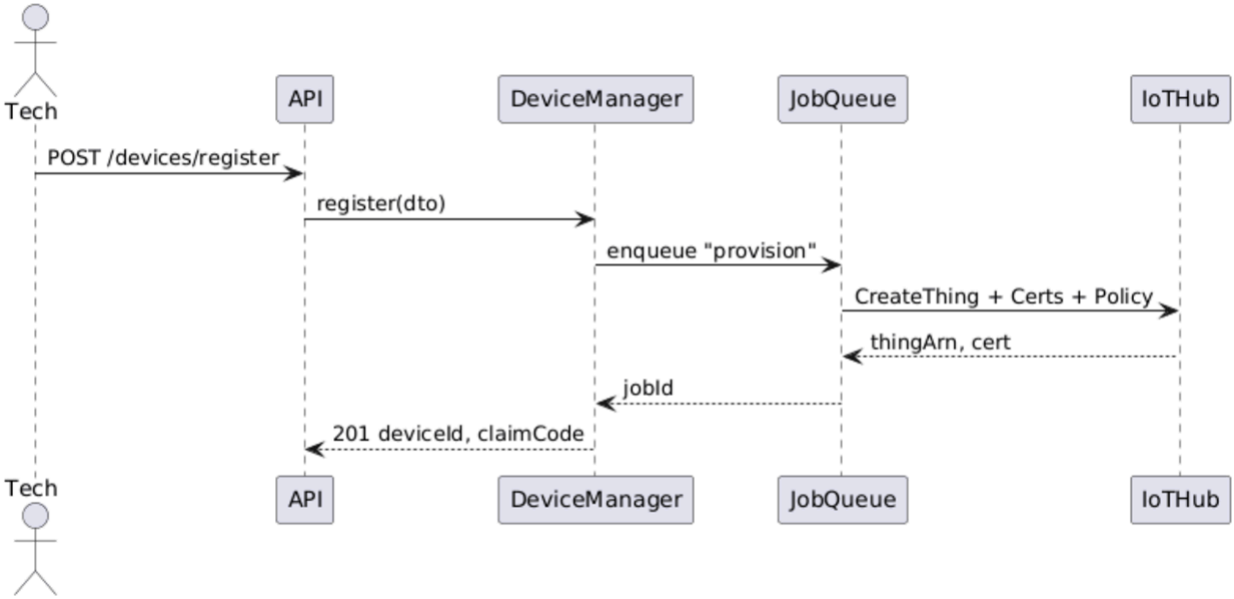
Class diagram

House owns Devices; each Device has Jobs (REBOOT/PTZ/PAIR/OTA) with status and timestamps.



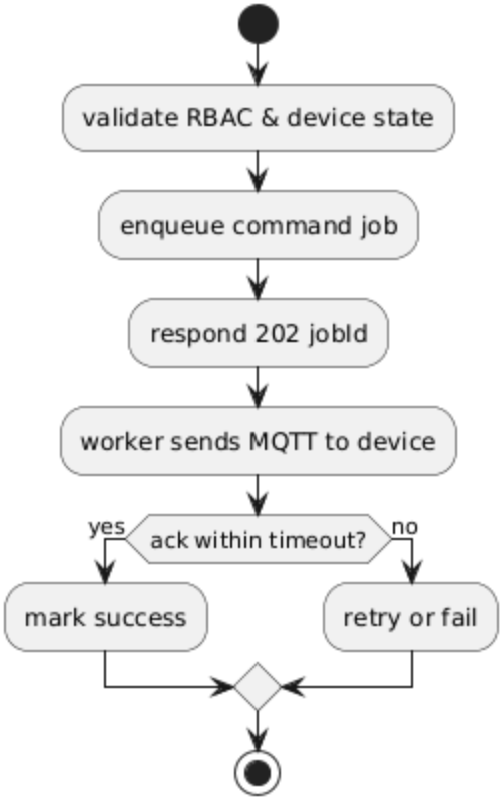
Sequence — onboarding

Register device → queue PAIR job → device reports progress → set ONLINE and push config.



Flowchart — command

Validate + throttle command → enqueue job → device executes → update status → audit.



DB (relational):

- device(id, serial, type, model, house_id, room, firmware, state, last_seen, created_at, updated_at)
- device_job(id, device_id, kind, args_json, status, created_at, updated_at)
  Indexes on (house_id,type,state) and (device_id, created_at).

Section 4: Business logic tables

Pairing acceptance

Condition	**Action**
Device claimed by other tenant	reject
Firmware < min_supported	force OTA then pair
Wi-Fi RSSI < threshold	warn + allow

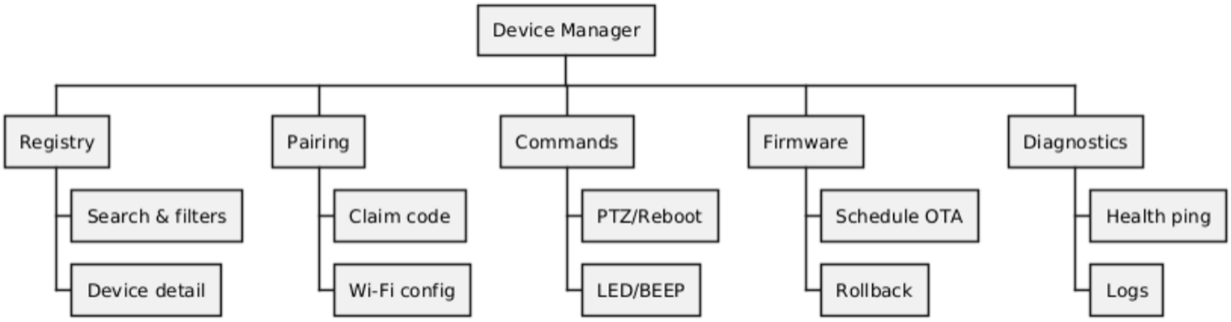
Command throttling

Kind	Limit
REBOOT	≤1 / 10 minutes
PTZ	≤30 / minute

Section 5: GUI design

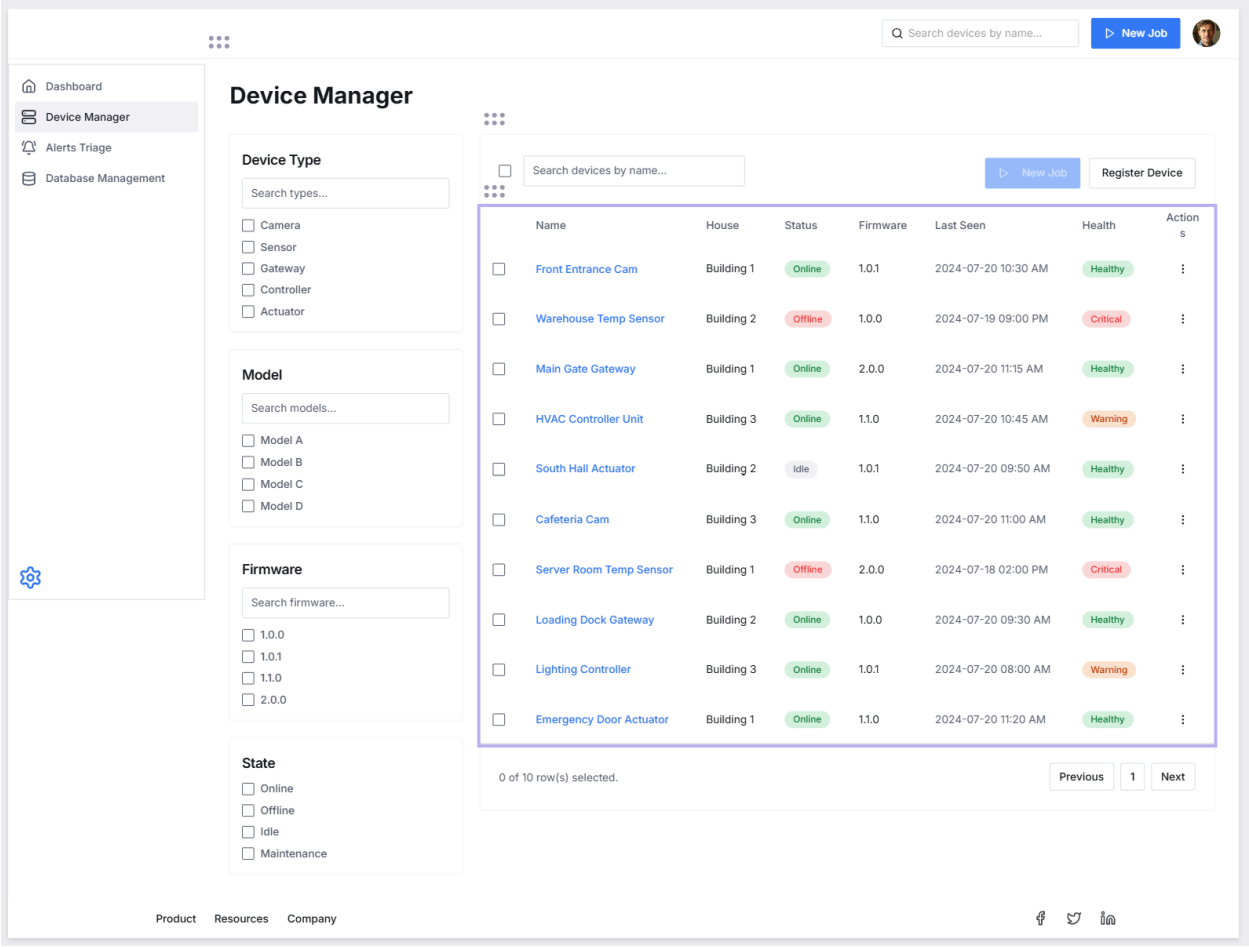
Style: Device table + detail drawer + job timeline.

WBS



Storyboard: search → open device → click “Pair” or “Command” → see job status → confirm success.

UI



Alert Tracking & Monitoring -- Nikhil Dupally

Section 1: Component overview

Purpose: real-time ingestion of AI/device events → deduplicate → score → create lifecycle-managed alerts → notify (WS/SMS/Email) → SLA & audit.
Objectives: real-time, severity scoring, policy-based escalation, multi-channel delivery, lifecycle (NEW→ACKED→RESOLVED→ARCHIVED), observability.
Scope: ingest APIs, alert store & query, notifications, timers, dedup, dashboards.
Usage: homeowners (ack/mute), IoT (device context), cloud staff (SLA, escalations).

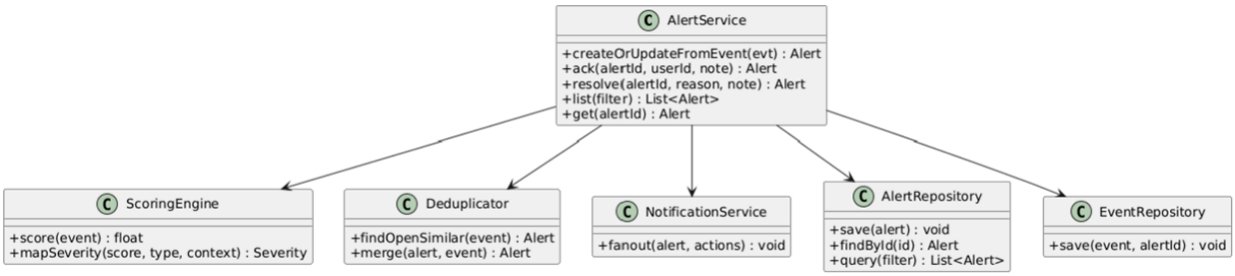
Section 2: APIs

- GET /api/v1/alerts?... list/filter
- GET /api/v1/alerts/{id} details + events + audit

- POST /api/v1/alerts/{id}/ack / resolve / mute
- GET /api/v1/alerts/stream (WebSocket)
- POST /api/v1/alerts/ingest/prediction and /ingest/device (internal)

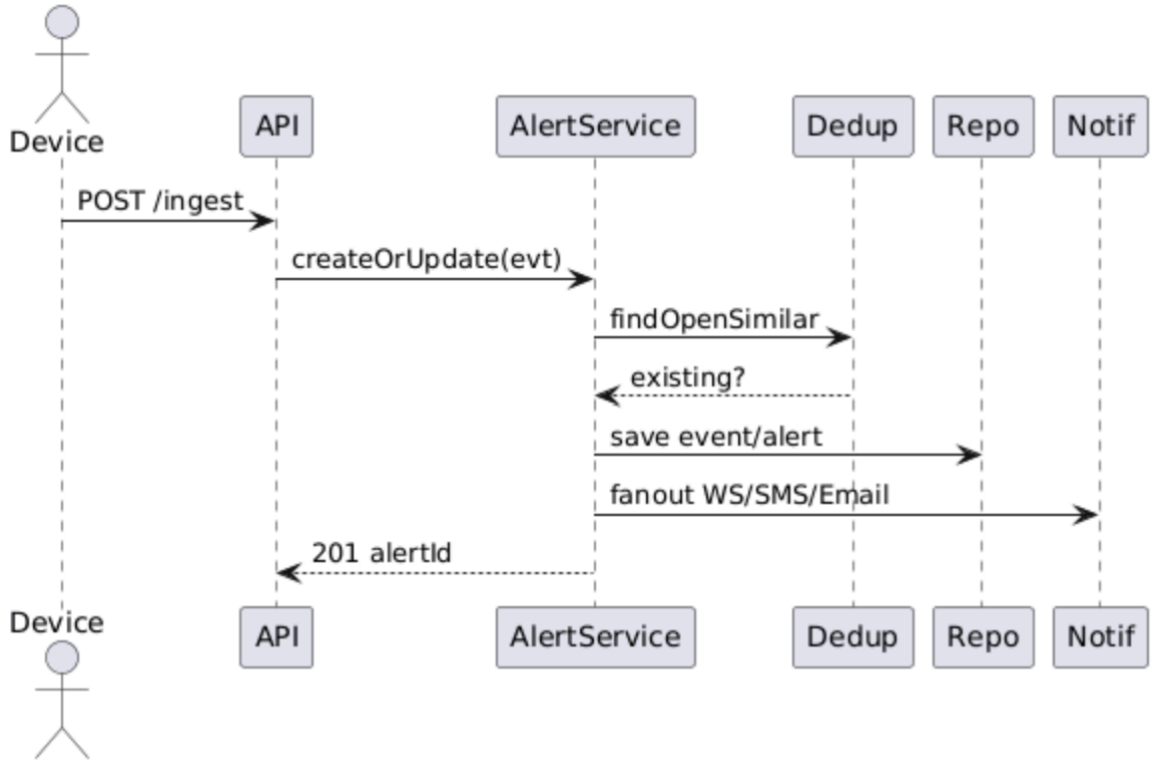
Section 3: Diagrams

Class Diagram:
Alert aggregates AlertEvents and Notifications; Policies influence behavior; Audit logs actions.



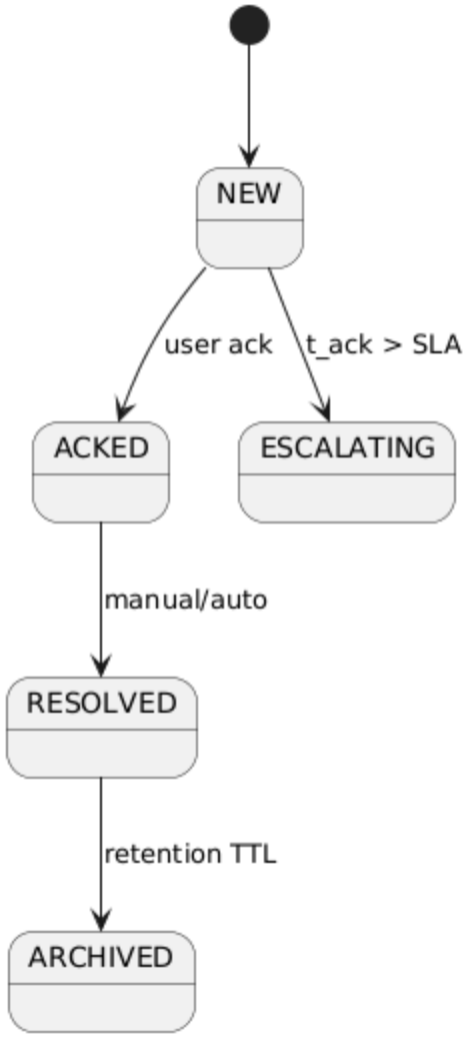
Sequence Diagram

Ingest event → compute dedupKey → upsert/open alert → route notifications → record audit.

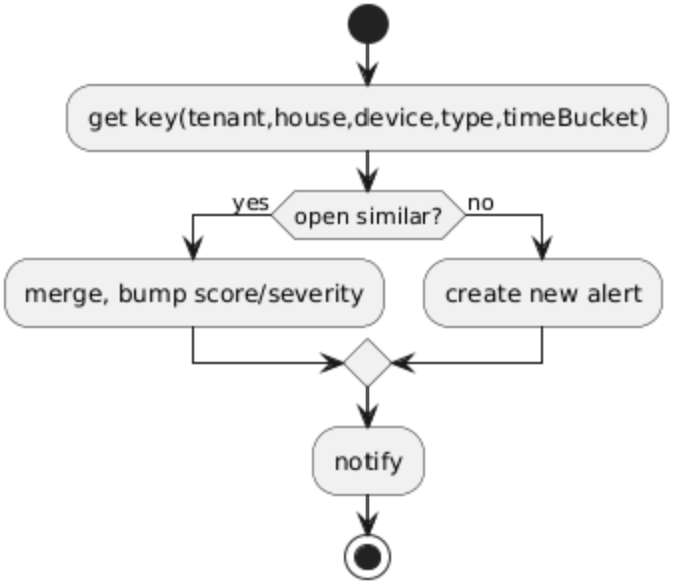


State — lifecycle

NEW → ACKED/RESOLVED → ARCHIVED via TTL or manual action



Flowchart



Section 4: Decision tables

Severity mapping (sample)

Type	Confidence	QuietHours	Severity
AUDIO.SCREAM	≥0.90	any	CRITICAL
MOTION.FALL	≥0.70	any	CRITICAL
GLASS_BREAK	≥0.80	any	HIGH
DEVICE.OFFLINE	n/a	good health	MEDIUM

Channel selection

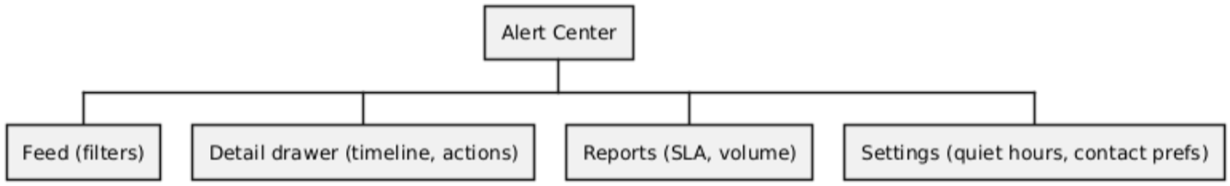
Severity	Not Acked Age	Channels
CRITICAL	immediate	WS + SMS + Email

HIGH	2–5 min	WS + Email
MEDIUM	≤10 min	WS
LOW	batched	Digest

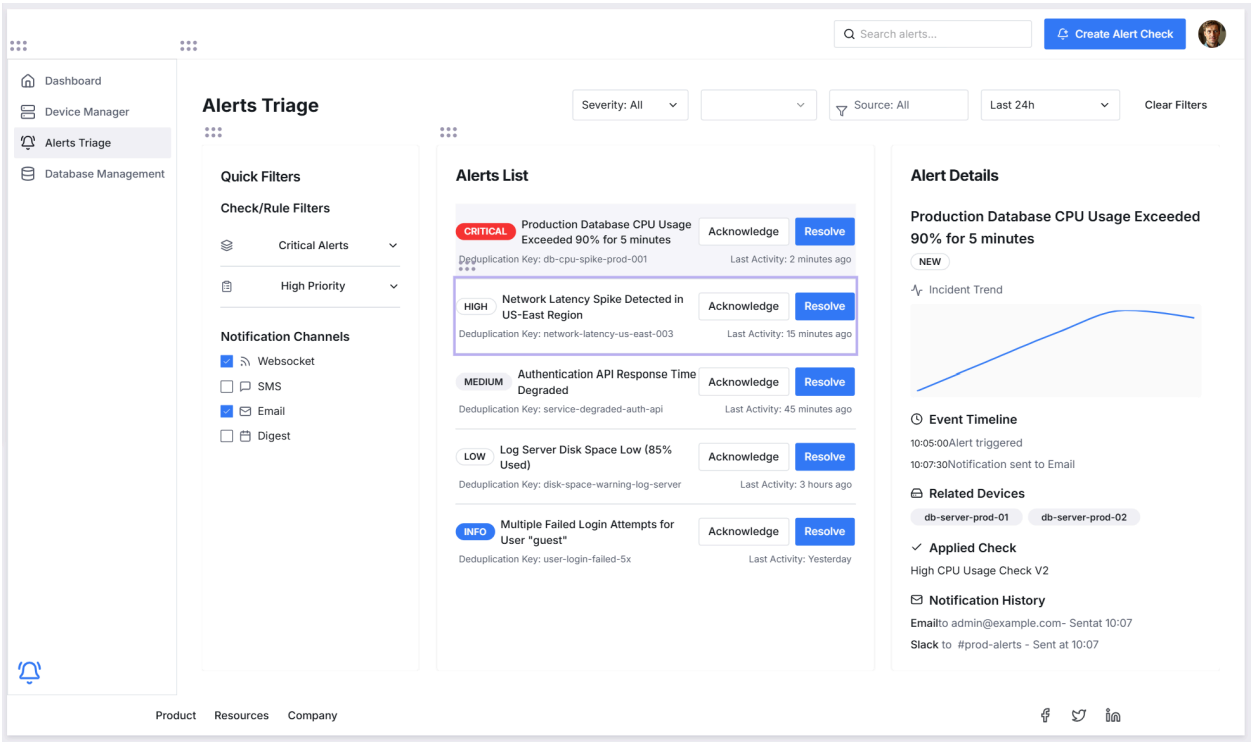
Section 5: GUI design

Style: alert list + filters; detail drawer with snapshot, timeline, SLA widget; real-time toasts.

WBS



UI



System Database Management – Divyasri Lakshmi Alekhya Nakka

Section 1: Component overview

Purpose: own the data architecture and operational data services: schemas, indexing/partitioning, lifecycle policies, backup/restore, data quality, and secure access.
Objectives:

- normalized relational core (tenants/houses/devices/alerts/audit)
- time-series/NoSQL for high-volume sensor/model outputs
- data lifecycle (hot → warm → archive), retention policies
- automated backups, PITR, restores, schema migrations, data catalog

Scope: DB design, migrations, DQ checks, CDC/ETL to lake (S3), RBAC at data level.
Usage: platform services and admins.

Section 2: Admin & service APIs

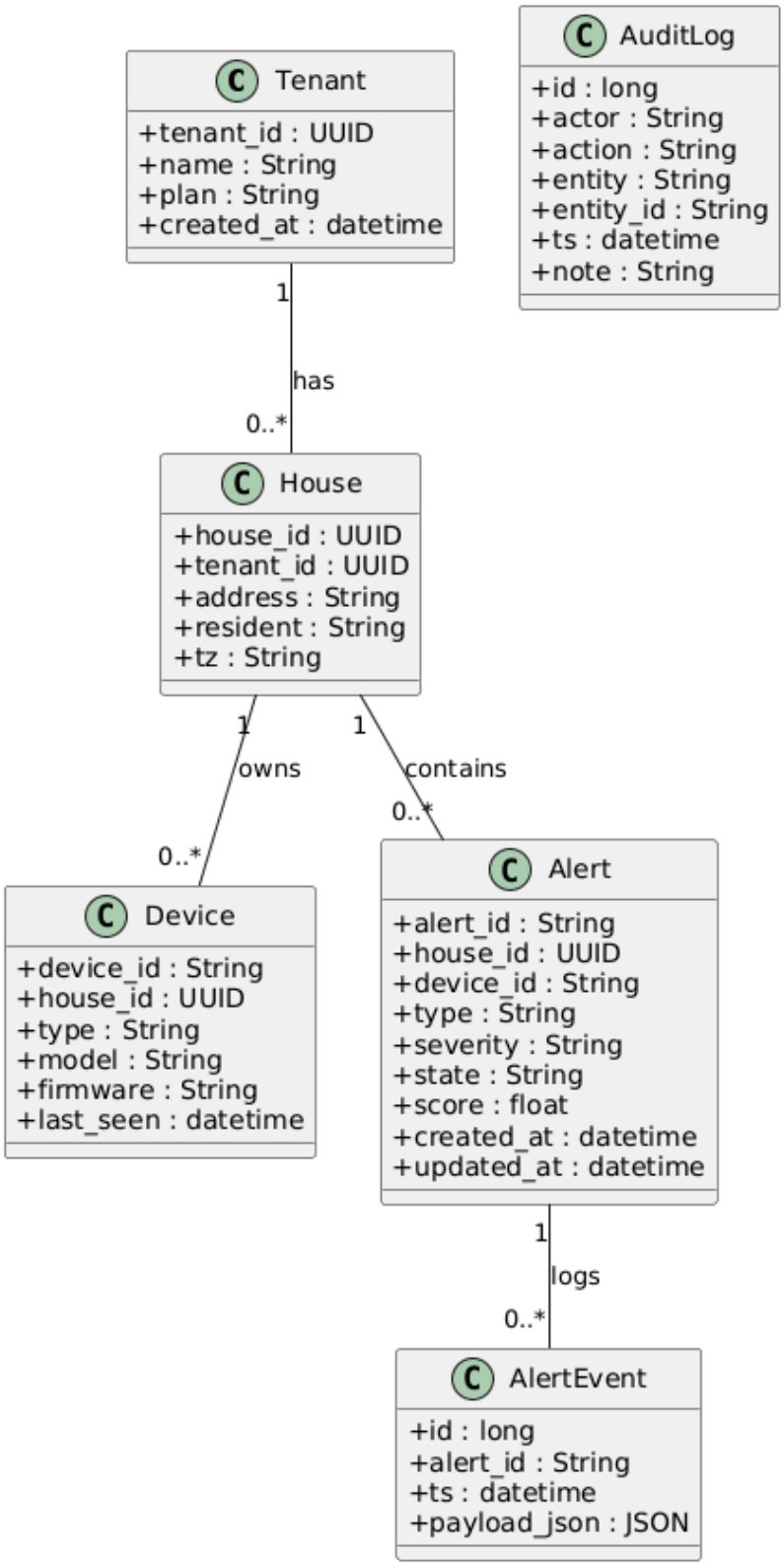
Base /api/v1/dbadmin (admin-only).

- GET /schemas → current schema versions & migration history
- POST /migrations/run {name} → apply idempotent migration
- GET /health → replica lag, disk pct, slow queries top-N
- GET /backups → schedules & latest snapshots
- POST /restore {snapshotId, target} → start restore job
- GET /retention-policies / PATCH /retention-policies
  Service endpoints (for other services):
- POST /ingest/sensor-batch (bulk write hints)
- POST /audit (append immutable audit event)

Section 3: Function, data & behavior

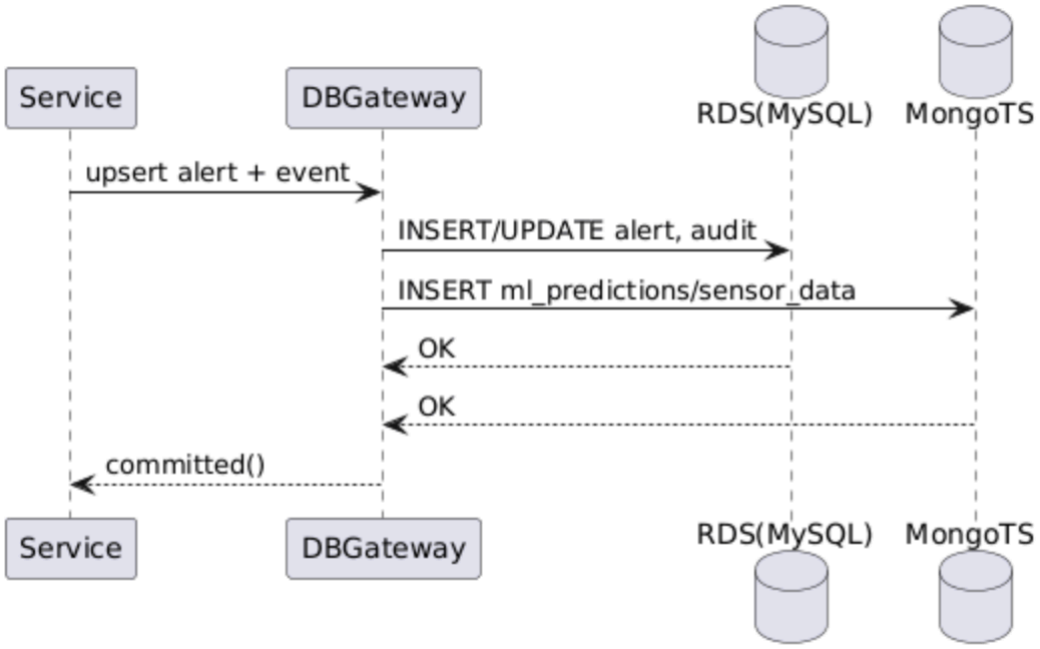
ER/Class diagram (core schema)

Tenant → House → Device/Alert core schema; events & audits normalized for history.



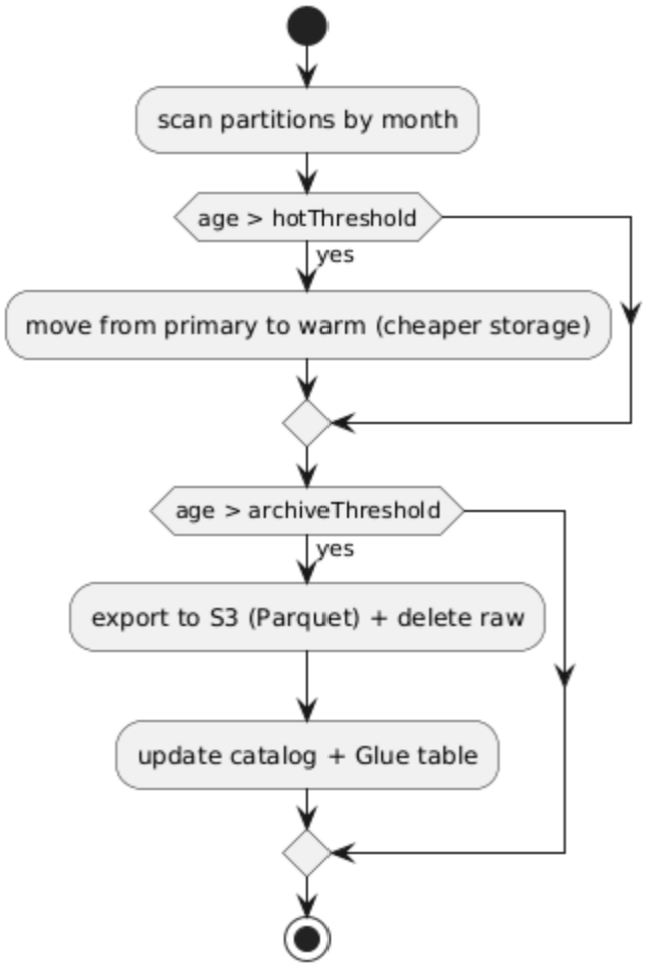
Sequence Diagram — service→DB

Service writes relational entities + timeseries; append audit; DBA monitors/migrates/backups.



Flowchart

Scheduled backups & retention → health checks → anomaly alerts → recovery runbook.



Schema snippets (write in report)

Relational:

```
CREATE TABLE tenant(...);
CREATE TABLE house(
  house_id VARCHAR(36) PRIMARY KEY,
  tenant_id VARCHAR(36) NOT NULL,
  address VARCHAR(200), tz VARCHAR(64),
  FOREIGN KEY (tenant_id) REFERENCES tenant(tenant_id)
);
-- device/alert/alert_event/audit_log as in diagram, with covering indexes:
```

CREATE INDEX ix_alert_house_state ON alert(house_id, state, updated_at DESC);

CREATE INDEX ix_device_house_type ON device(house_id, type);

Time-series:

db.sensor_data.createIndex({tenant_id:1, house_id:1, ts:1});

db.sensor_data.createIndex({ts:1},{expireAfterSeconds:60*60*24*90});

Data quality checks: not-nulls, FK integrity, valid enum ranges, ts monotonicity, duplicate alert dedup keys, nightly anomaly checks.
Backups: daily snapshots + PITR; restore tests monthly.
Security: row-level scoping by tenant_id, KMS at rest, TLS in transit, masked PII in non-prod.

Section 4: Business logic (decision tables)

Retention policy

Data type	**Hot (RDS/Mongo)**	**Warm (partitioned)**	**Archive (S3 Parquet)**
Alerts & audits	90 days	9–12 months	2–5 years
Raw sensor data	30 days	2–3 months	6–24 months

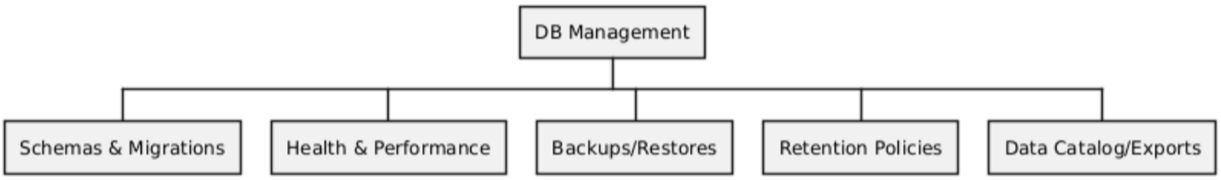
Migration gate

Check	**Block deploy?**
New columns nullable?	No
Backfill completed?	Yes if no
Index created online?	Yes if no

Section 5: DB Admin GUI

Style: admin-only console; tabs for Schemas • Health • Backups • Retention • Catalog.

WBS



Storyboard: open DB Console → view health (replica lag) → schedule backup → adjust retention → view catalog table preview → export sample.

UI