

**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

	<b>1</b>
<b>System Dashboard -- Vijaya Sharavan Reddy Baddam</b>	3
Section 1: Component overview	3
Section 2: Component APIs (REST)	3
Section 3: Function & data design, behavior	3
Section 4: Business logic (decision tables)	5
Section 5: GUI design	5
<b>Device Manager -- Venkata Gowtham Jalam</b>	6
Section 1: Component overview	6
Section 3: Function & data design	7
Section 4: Business logic tables	9
Section 5: GUI design	9
<b>Alert Tracking &amp; Monitoring -- Nikhil Dupally</b>	10
Section 1: Component overview	10
Section 2: APIs	10
Section 3: Diagrams	11
Section 4: Decision tables	13
Section 5: GUI design	14
<b>System Database Management – Divyasri Lakshmi Alekhya Nakka</b>	15
Section 1: Component overview	15
Section 2: Admin & service APIs	15
Section 3: Function, data & behavior	15
Section 4: Business logic (decision tables)	19
Section 5: DB Admin GUI	<b>19</b>

# 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)

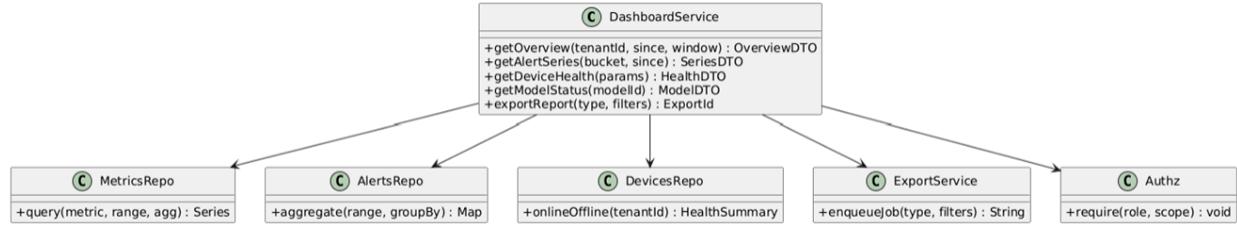
Base path /api/v1/dashboard (JWT, role scoped, 60 rpm/user).

- 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

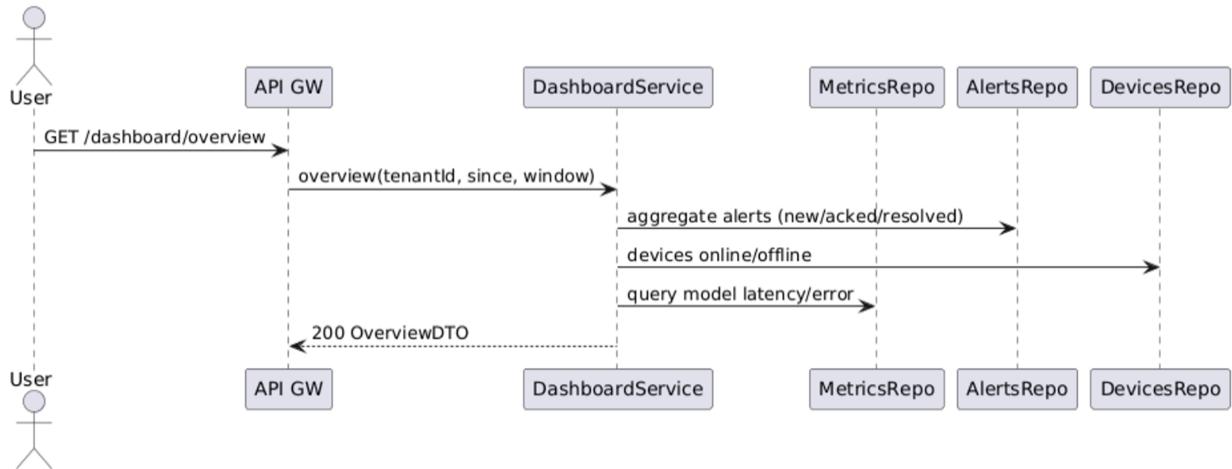
## Class diagram

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



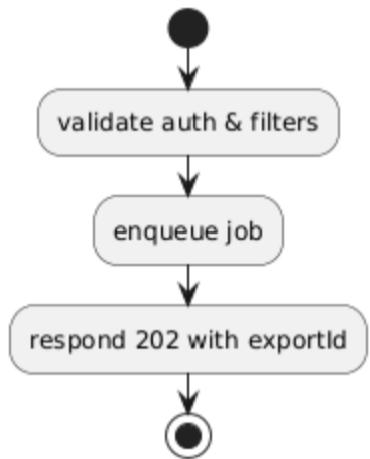
## Sequence — overview KPI fetch

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)

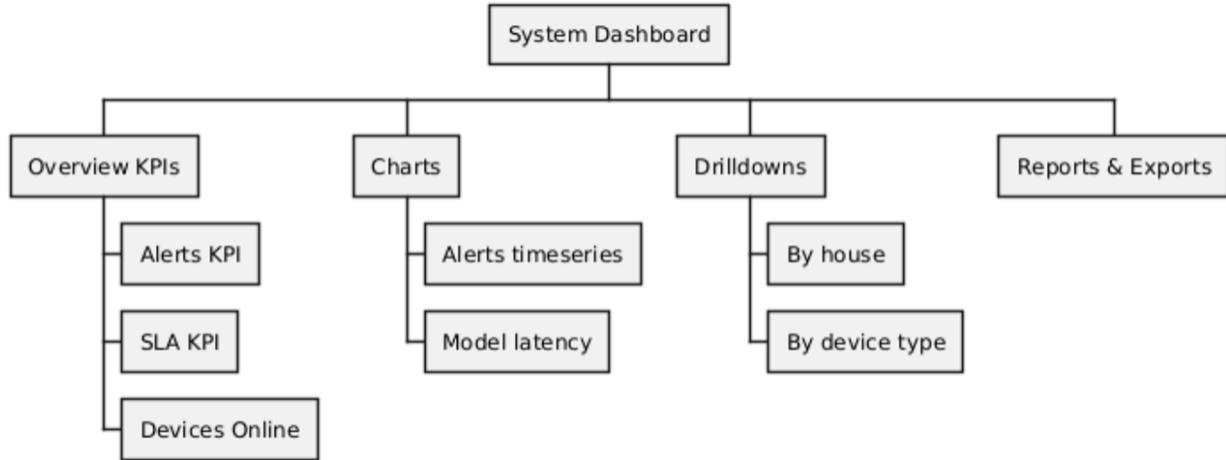
### SLA coloring

KPI	Thresholds	Color
Ack SLA	$\geq 95\% = \text{Green}$ ; $80\text{--}95\% = \text{Amber}$ ; $< 80\% = \text{Red}$	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**

[Dashboard](#)

[Device Manager](#)

[Alerts Triage](#)

[Database Management](#)

[Export CSV](#)
[Create Alert Policy](#)

Total Devices  
**1,245**  
↑ 1.2% ↑

Online %  
**98.5 %**  
↑ 0.1% ↑

Open Alerts  
**147**  
↓ 5% ↓

SLA Compliance  
**99.9 %**  
↑ 0.01% ↑

Time Range :

Tenant :

### 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

House	Devices
House A	220
House B	180
House C	360
House D	100
House E	270

**Alert Severity Breakdown**

Distribution of active alerts

Severity	Percentage
Critical	~10%
Major	~60%
Minor	~30%

● Critical ● Major ● Minor

**Recent Alerts**

Latest incidents across your platform

Severity	Source	Time	Status	Actions
Critical	Device-X	2 min ago	<span>Open</span>	<a href="#">View</a>
Major	Database-S	15 min ago	<span>Acknowledged</span>	<a href="#">View</a>
Minor	Network-R	30 min ago	<span>Open</span>	<a href="#">View</a>
Critical	Device-Y	1 hour ago	<span>Open</span>	<a href="#">View</a>
Major	Platform-Z	2 hours ago	<span>Resolved</span>	<a href="#">View</a>

[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)

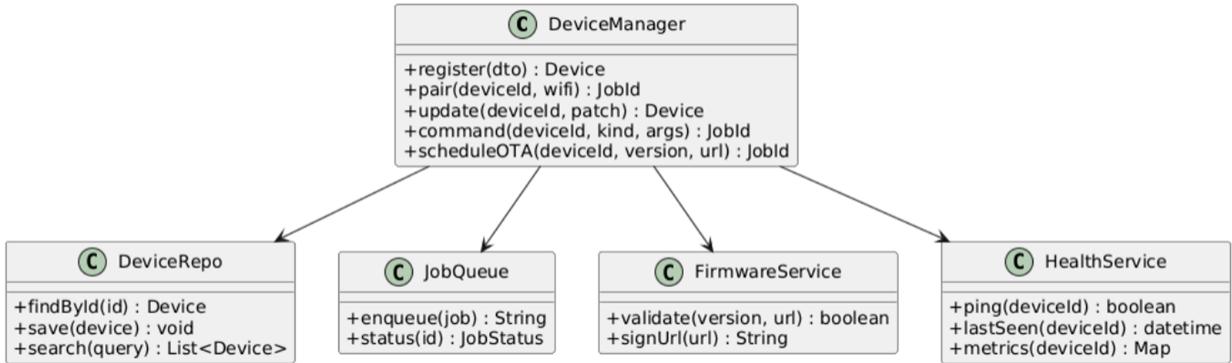
Base /api/v1/devices (JWT + RBAC; device commands are async).

- 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

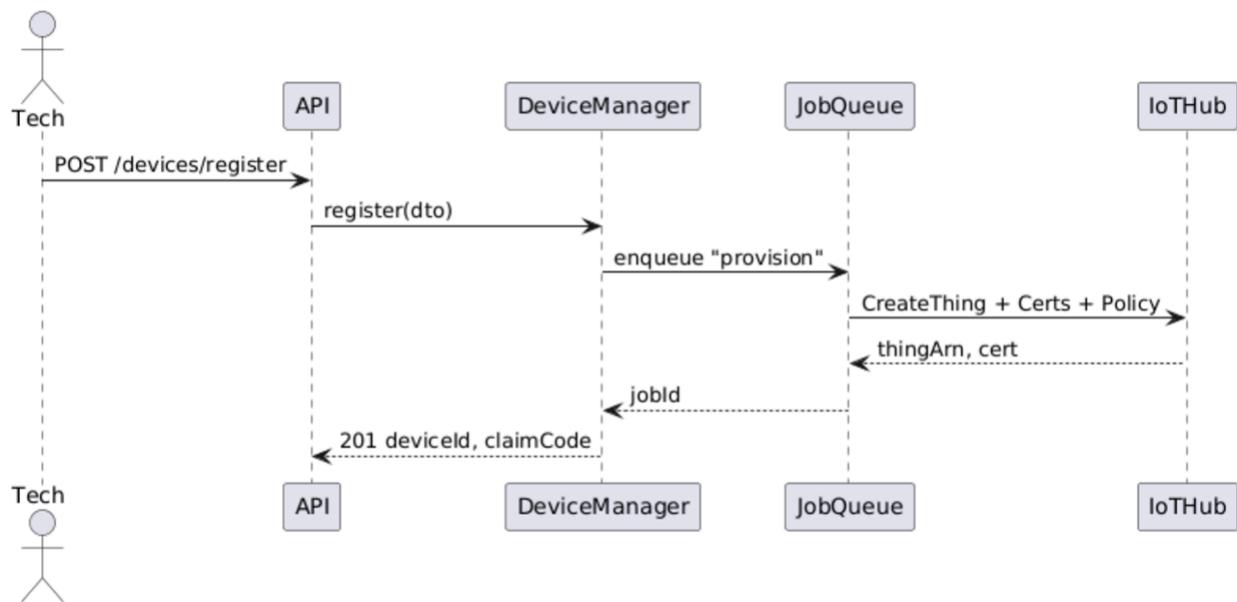
### Class diagram

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



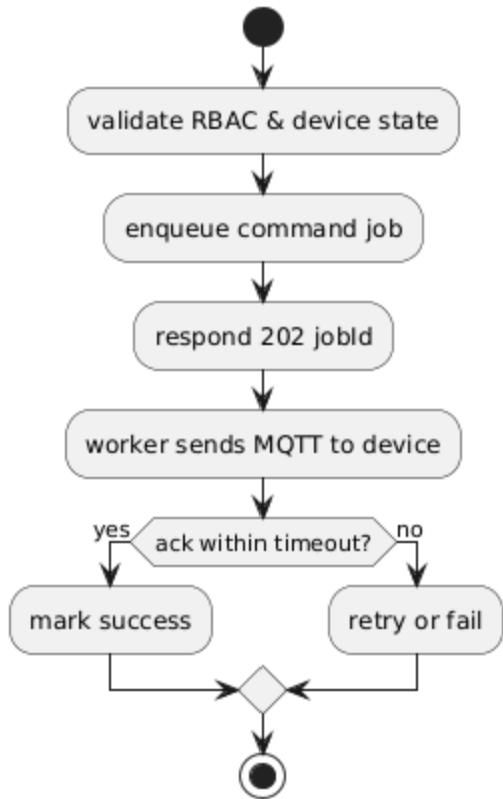
### Sequence — onboarding

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

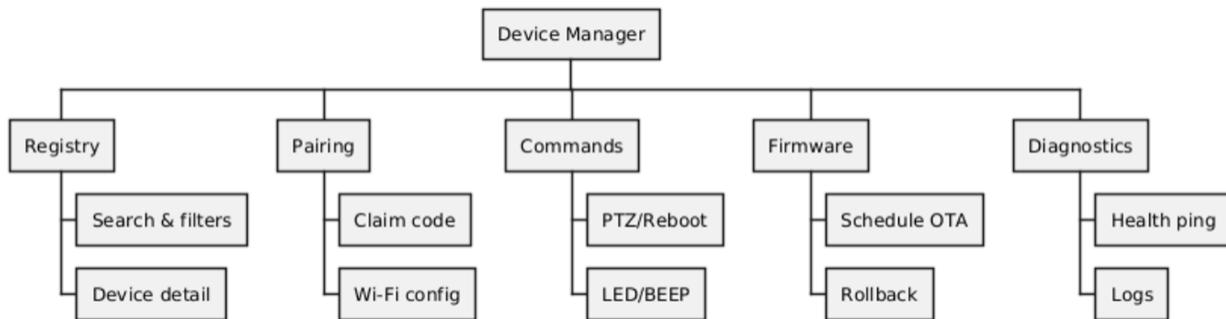
## Command throttling

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

The screenshot shows a Device Manager interface with the following details:

Name	House	Status	Firmware	Last Seen	Health	Actions
Front Entrance Cam	Building 1	Online	1.0.1	2024-07-20 10:30 AM	Healthy	⋮
Warehouse Temp Sensor	Building 2	Offline	1.0.0	2024-07-19 09:00 PM	Critical	⋮
Main Gate Gateway	Building 1	Online	2.0.0	2024-07-20 11:15 AM	Healthy	⋮
HVAC Controller Unit	Building 3	Online	1.1.0	2024-07-20 10:45 AM	Warning	⋮
South Hall Actuator	Building 2	Idle	1.0.1	2024-07-20 09:50 AM	Healthy	⋮
Cafeteria Cam	Building 3	Online	1.1.0	2024-07-20 11:00 AM	Healthy	⋮
Server Room Temp Sensor	Building 1	Offline	2.0.0	2024-07-18 02:00 PM	Critical	⋮
Loading Dock Gateway	Building 2	Online	1.0.0	2024-07-20 09:30 AM	Healthy	⋮
Lighting Controller	Building 3	Online	1.0.1	2024-07-20 08:00 AM	Warning	⋮
Emergency Door Actuator	Building 1	Online	1.1.0	2024-07-20 11:20 AM	Healthy	⋮

0 of 10 row(s) selected.

## 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

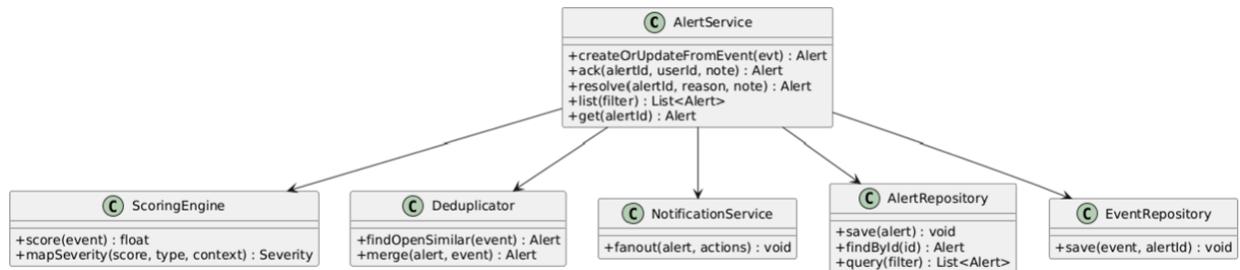
- GET /api/v1/alerts?... list/filter
- GET /api/v1/alerts/{id} details + events + audit

- 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

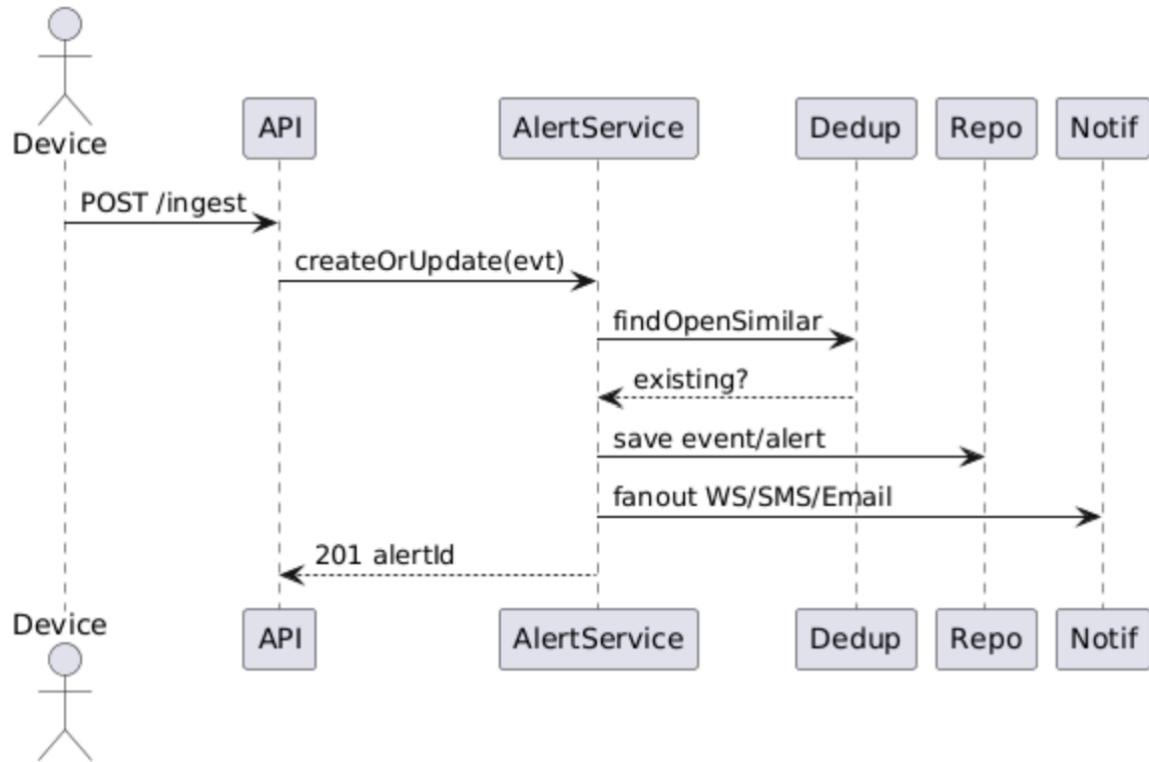
### Class Diagram:

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



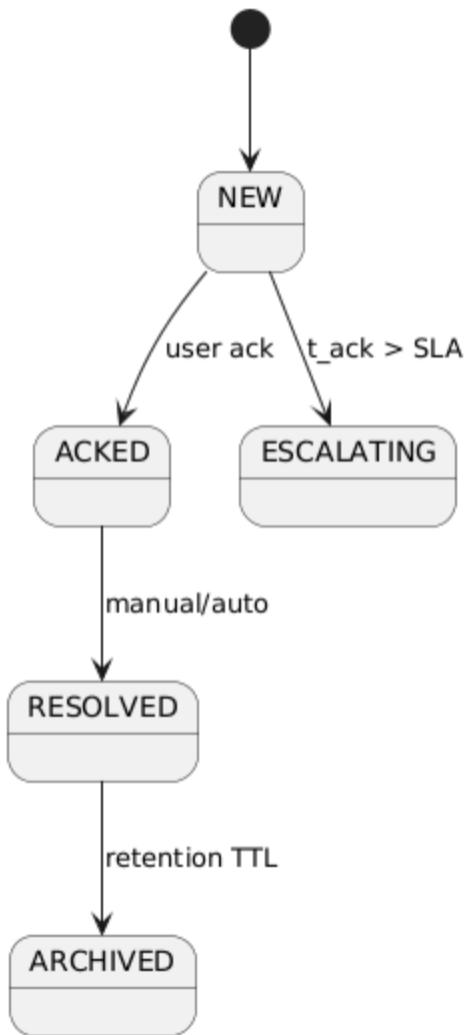
### Sequence Diagram

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

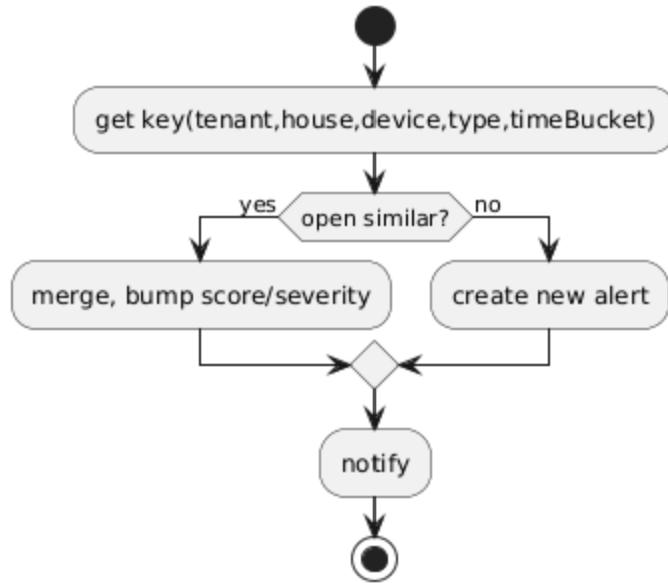


### State — lifecycle

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



**Flowchart**



## Section 4: Decision tables

### Severity mapping (sample)

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

### Channel selection

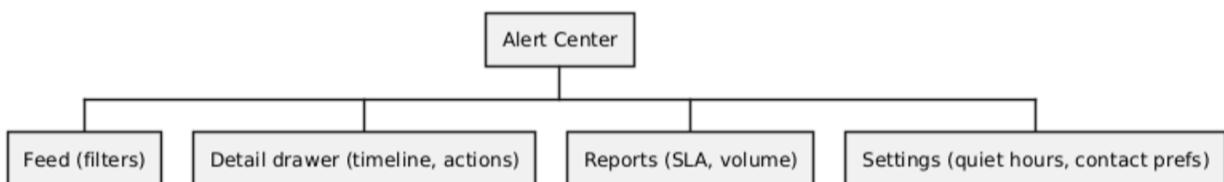
Severity	Not Acked Age	Channels
CRITICAL	immediate	WS + SMS + Email

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

The screenshot shows the 'Alerts Triage' section of the Alert Center. On the left, there's a sidebar with navigation links: Dashboard, Device Manager, Alerts Triage (which is selected and highlighted in grey), and Database Management. Below these are 'Quick Filters' for Critical Alerts and High Priority, and 'Notification Channels' for Websocket, SMS, Email, and Digest. The main area has a search bar at the top right with filters for Severity (All), Source (All), and Last 24h, along with a 'Clear Filters' button. The 'Alerts List' section displays four alerts with details like severity, message, deduplication key, and last activity. Each alert has 'Acknowledge' and 'Resolve' buttons. To the right, the 'Alert Details' section provides more information about the first alert (Production Database CPU Usage Exceeded 90% for 5 minutes) including an incident trend graph, event timeline, related devices (db-server-prod-01, db-server-prod-02), applied checks (High CPU Usage Check V2), and notification history (Email to admin@example.com sent at 10:07, Slack to #prod-alerts at 10:07).

# System Database Management – Divyasri Lakshmi Alekhyा 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

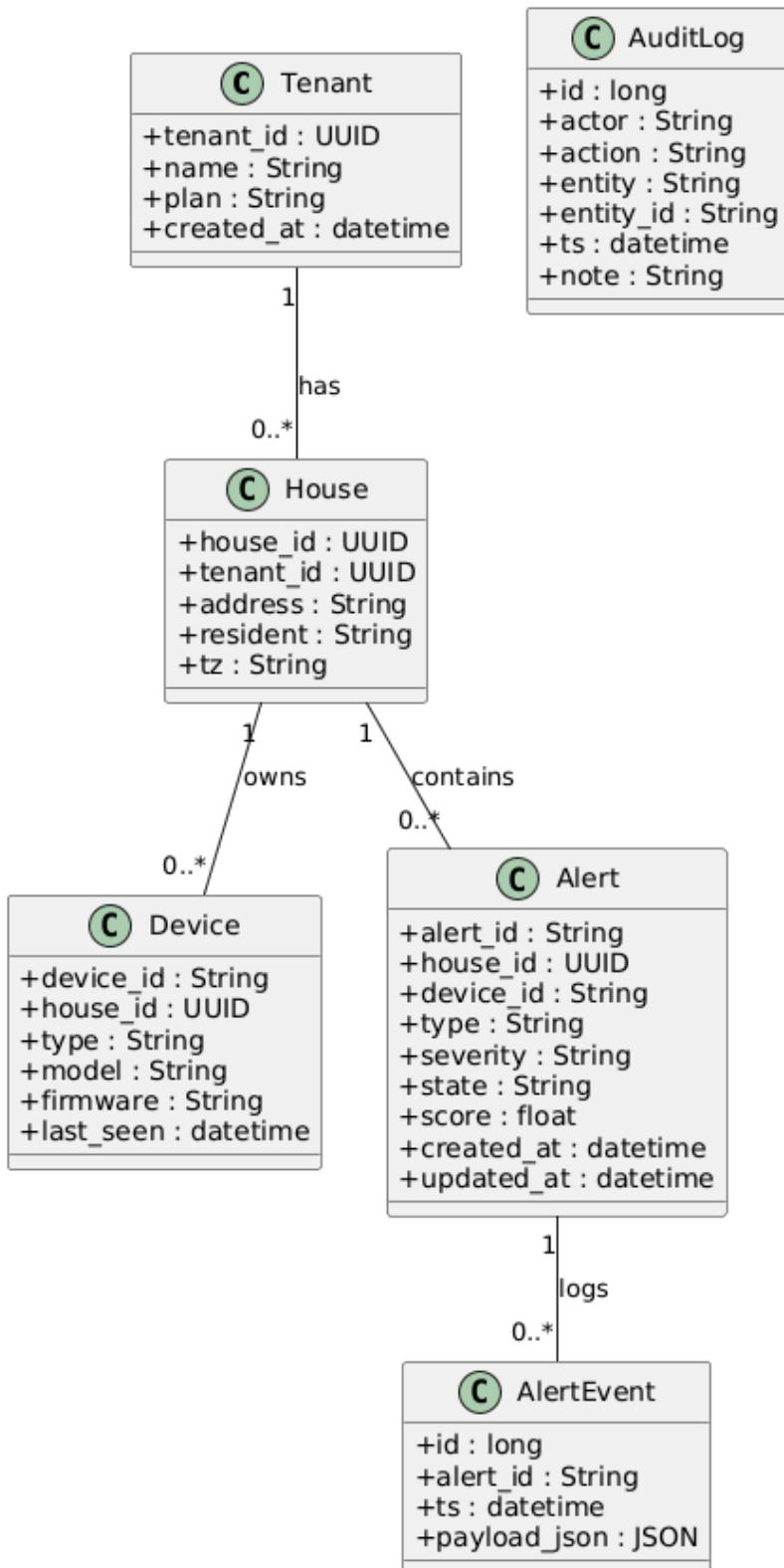
**Base /api/v1/dbadmin (admin-only).**

- 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

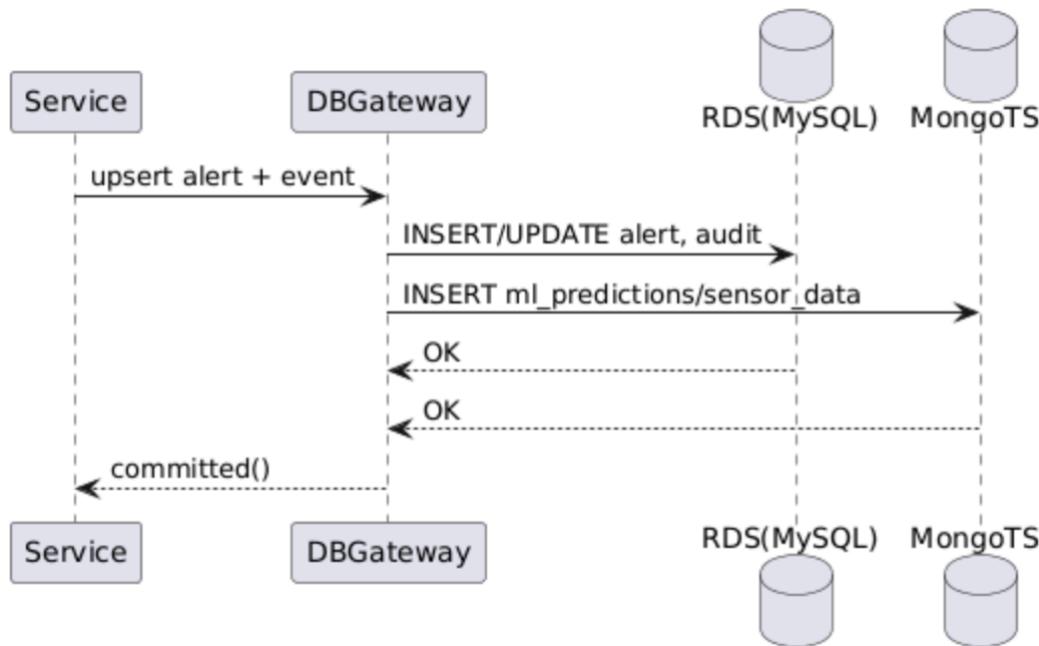
**ER/Class diagram (core schema)**

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



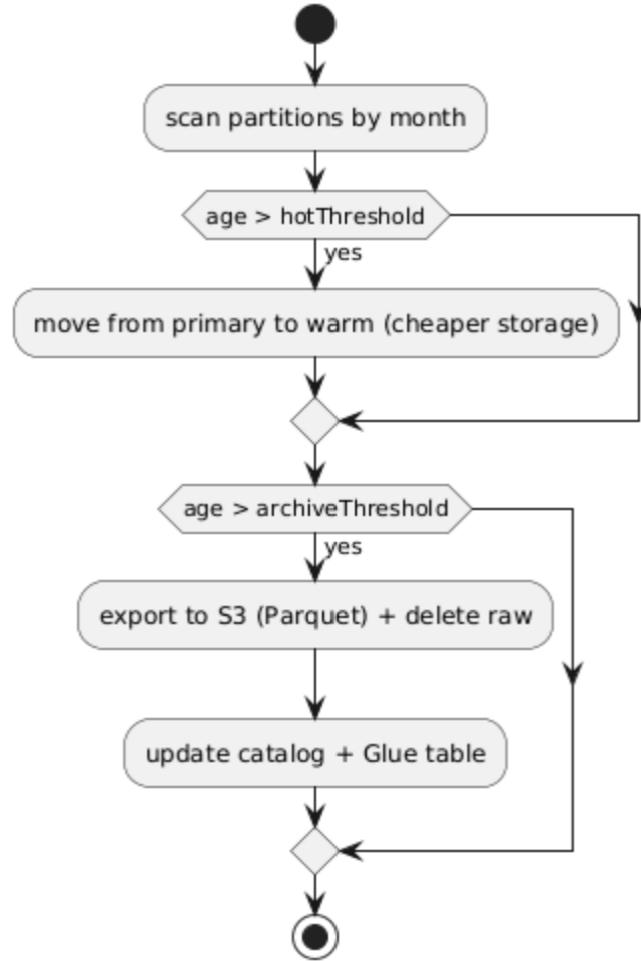
Sequence Diagram — service→DB

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

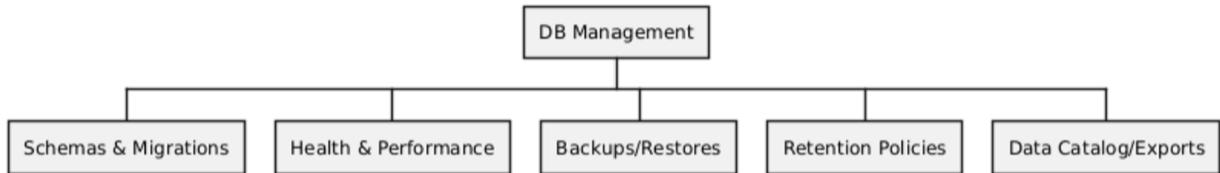
#### Migration gate

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

The screenshot shows a Database Management interface. On the left is a sidebar with navigation links: Dashboard, Device Manager, Alerts Triage, and Database Management (which is selected). Below the sidebar is a "Schema Browser" section with dropdown menus for Tenant Schema, House Schema, Device Schema, Alert Schema, AlertEvent Schema, and Audit Schema. The main content area is titled "Database Management" and includes tabs for Overview, Tables, Migrations, Backups, and Health. The Overview tab is active, displaying four key performance metrics: Replication Lag (5 ms, +1.5% from last month), Storage Utilization (78.2%, Growing by 0.5% weekly), Slow Query Count (12, Decreased by 20% in 24h), and Retention Status (90 days, Configured for 3 months). The bottom of the screen features a footer with links for Product, Resources, Company, and icons for Home, Help, Log Out, and Refresh.