intelbyte

Designing a Scalable Factory Equipment Health and Maintenance Platform

Architecture, Data Model, and Algorithms

Table of Contents

1 Executive Summary

This document presents the technical design for a Factory Equipment Health and Maintenance Platform developed for Intelbyte. The objective of the platform is to provide clear, real-time visibility into the health and operational status of equipment across multiple manufacturing plants, while preserving a complete historical record of machine telemetry, operator inputs, and maintenance activity for analysis, reporting, and audit purposes.

The proposed solution uses a scalable, event-driven data architecture that separates high-volume telemetry ingestion from day-to-day operational decision-making. Machine sensor readings such as temperature, vibration, throughput, and fault codes are captured as timestamped events for long-term retention, while a derived current equipment state is maintained for each machine. This enables supervisors to view a single, up-to-date record per asset without querying large historical datasets, even as data volumes grow.

The platform supports multiple plants, production lines, and machines, along with operator shift reports, manual inspections, and status overrides. Maintenance requests are generated automatically based on configurable thresholds, fault patterns, and operator-reported issues. A machine health scoring process combines recent telemetry signals and fault history into a single, easy-to-interpret score, while a clear precedence model resolves the current machine status across telemetry, operator inputs, and active work orders.

Overall, the design is practical, transparent, and scalable. It delivers immediate value for day-to-day operations and supports timely maintenance decisions, while leaving room to add features such as predictive maintenance, deeper analysis, and AI based insights as the platform grows.

2 Problem Context and Objectives

Manufacturing environments operate large numbers of machines across multiple plants and production lines, generating continuous streams of sensor data alongside operator reports and inspections. While this information is essential for monitoring equipment condition, its volume and fragmentation make it difficult to use effectively for day-to-day operational decisions.

A central challenge is maintaining a clear, up to date view of current equipment status while preserving complete historical data for analysis, reporting, and audits. Supervisors require a simple snapshot of machine health and status, whereas engineers and analysts need access to detailed historical records to identify trends, recurring issues, and long term performance patterns. Without a unified approach, teams often rely on manual processes, disconnected data sources, and reactive maintenance practices.

The objective of this platform is to address these challenges by providing a unified, scalable foundation for equipment health and maintenance management. At a high level, the platform is designed to:

- Provide a single, reliable view of the current status and health of each machine
- Ingest and retain high volume telemetry data for long term analysis and audits
- Capture operator reports, inspections, and status changes alongside sensor data
- Automatically trigger maintenance requests based on thresholds, fault patterns, and reported issues
- Support growth across plants, machines, and data volumes without major redesign

3 Requirements Analysis

The following requirements are defined based on the scenario provided, which assumes the availability of machine telemetry, operator inputs, and maintenance workflows typical of modern manufacturing environments.

3.1 Core Functional Requirements

These define the essential capabilities the platform provides to support equipment monitoring, maintenance, and operational decision-making. They are organized by functional area for clarity.

3.1.a Asset and Equipment Management

- Support multiple plants, production lines, and machines
- Maintain a clear hierarchical relationship between plants, lines, and machines

3.1.b Telemetry and Data Ingestion

- Ingest continuous machine telemetry, including temperature, vibration, throughput, and fault codes
- Associate each telemetry record with a machine identifier and timestamp
- Retain telemetry data as a complete historical record for analysis, reporting, and audits

3.1.c Operator and User Inputs

- Allow operators to log shift reports, inspection results, and issue descriptions
- Record manual machine status updates with user attribution
- Store operator comments and inspection notes alongside machine and telemetry data

3.1.d Maintenance Management

- Automatically create maintenance requests based on configurable thresholds, fault patterns, and operator reported issues
- Track maintenance requests through their lifecycle, including status updates and resolution details
- Link maintenance history to machines and triggering events

3.1.e Health and Status Monitoring

- Calculate a machine health score using recent telemetry signals and fault history
- Determine the current machine status by reconciling telemetry data, operator inputs, and active maintenance work orders
- Provide a Current Equipment Status view showing one row per machine using the most recent available data

3.1.f Reporting and Dashboards

- Provide dashboards for supervisors to monitor equipment health, status, and maintenance activity
- Support filtering by plant, production line, machine, status, and health indicators

3.2 Operational Constraints and Quality Considerations

Beyond core functionality, the platform is designed to operate reliably in a production environment where data volume and operational complexity increase over time. It is expected to scale to thousands of machines generating high-frequency telemetry without requiring major redesign or manual intervention.

Operational views used by supervisors and maintenance teams must remain highly available and reflect the latest known state of each machine in near real-time. To support audits, investigations, and long-term analysis, all telemetry and operational events are preserved as immutable historical records. The platform enforces role based access control to ensure users can only view or perform actions appropriate to their role, protecting sensitive operational data and workflows. System components are loosely coupled so ingestion, processing, analytics, and user facing features can be updated or extended independently, supporting continuous improvement without disrupting ongoing operations.

4 System Architecture

The platform follows an event-driven architecture that separates high-volume telemetry ingestion from operational workflows and user facing views. Machine telemetry is captured as append-only event data to support long-term retention, analysis, and auditability. In parallel, a derived current equipment state is maintained to enable fast operational queries that present one up-to-date record per machine.

At a high level, the system consists of a web application for operators and supervisors, backend services that manage operational workflows and business rules, a telemetry ingestion path optimized for scale, and a database layer that stores both historical telemetry and current machine state. Maintenance requests are generated through configurable rules and recorded as work items linked to the machines and events that triggered them.

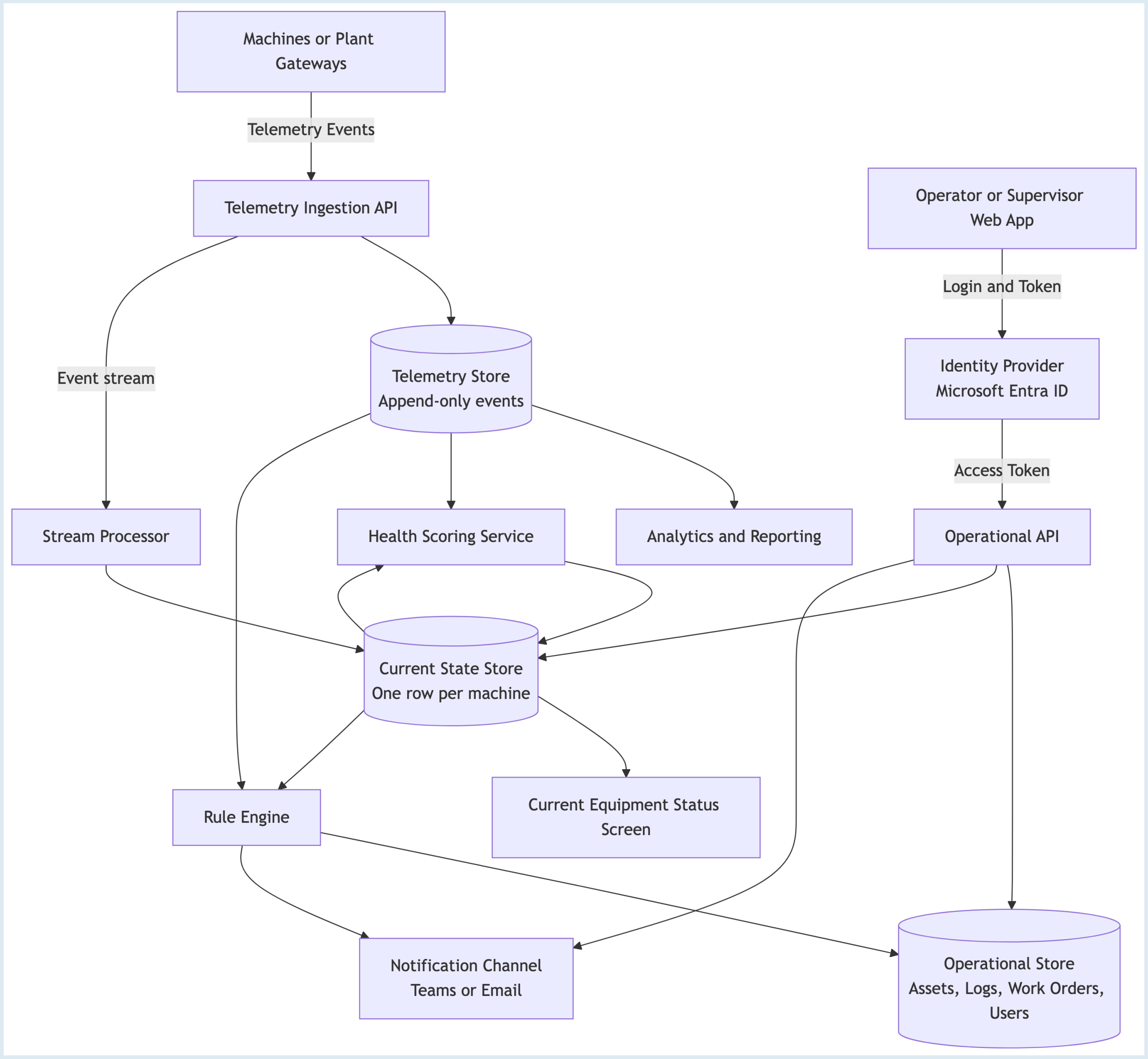
Machines or Plant Gateways
Telemetry Events
Telemetry Ingestion API
Operator or Supervisor Web App
Login and Token
Event stream
Telemetry Store Append-only events
Identity Provider Microsoft Entra ID
Access Token
Stream Processor
Health Scoring Service
Analytics and Reporting
Operational API
Current State Store One row per machine
Rule Engine
Current Equipment Status Screen
Notification Channel Teams or Email
Operational Store Assets, Logs, Work Orders, Users

Figure 1: System architecture and data flow

4.1 Components and Functions

4.1.a Frontend (Operator and Supervisor Interfaces)

- Operator and Supervisor Web App (Power Apps): provides a unified interface for operators, supervisors, and maintenance teams to view current equipment status, review machine details, log shift reports and inspections, submit maintenance requests, and review, prioritize, assign, and track maintenance work orders.
- Analytics Dashboards (Power BI): visual dashboards showing equipment health, status, and maintenance activity by plant, production line, and machine

4.1.b Backend Services and APIs (Application and Processing Layer)

- Operational API (Azure Functions): manages core business operations including plant and machine metadata, operator inputs, status updates, and maintenance workflows
- Telemetry Ingestion API (Azure IoT Hub): receives sensor data from machines or plant gateways and checks that required information is present
- Event Stream (Azure Event Hubs): temporarily buffers incoming telemetry to allow the system to scale and continue receiving data even during peak loads
- Stream Processor (Azure Stream Analytics): continuously processes incoming telemetry, determines the current condition of each machine using predefined rules, and keeps the operational view up to date.
- Rule Engine (Azure Functions): evaluates business rules against machine state and history to decide when maintenance actions are required, creating work orders and triggering notifications.
- Health Scoring Service (Azure Functions): calculates a simple health score for each machine using recent telemetry and fault history

4.1.c Data Layer (Data Storage and Access)

- Operational Store (Azure SQL Database): stores structured business data including assets, users, operator reports, and maintenance records
- Telemetry Store (Azure Data Explorer): stores all raw telemetry data as a long-term historical record for analysis and audits
- Current State Store (Azure SQL Database): stores the latest status and health indicators for each machine to support fast operational views

4.1.d Integrations (Access and Notifications)

- Identity Provider (Microsoft Entra ID): manages user login, roles, and access permissions
- Notification Channel (Power Automate with Teams or Email): sends alerts when maintenance requests are created or critical issues are detected

4.2 Data Flow and Communication

Telemetry and operator inputs follow different paths, but converge in the current equipment state used for operational dashboards.

4.2.a Telemetry ingestion flow

1. Machines or plant gateways send telemetry events to the Telemetry Ingestion API
2. Events are validated and written to the Telemetry Store as immutable, append-only records
3. A stream processor consumes telemetry events and updates the Current State Store for the affected machine
4. The Health Scoring Service recalculates machine health scores based on new data or on a defined schedule
5. The Rule Engine evaluates thresholds and fault patterns and creates maintenance requests when conditions are met

4.2.b Operator input flow

1. Operators submit shift logs, inspections, comments, or manual status updates through the web application
2. The Operational API records these inputs in the Operational Store with user attribution
3. Inputs that affect machine status are reflected in the Current State Store
4. Operator-reported issues may directly trigger maintenance requests or contribute to rule evaluation

4.2.c Operational dashboard flow

- The Current Equipment Status screen queries the Current State Store to retrieve one row per machine for fast operational monitoring.
- Historical analysis and audits query the Telemetry Store and join with operational data as needed for reporting and compliance.

4.3 Architecture Pattern and Rationale

The architecture follows an event-driven pattern with a derived current-state view to handle high-frequency telemetry that grows rapidly over time. Telemetry is stored as immutable events to preserve a complete historical record for analysis and audits, while a separate current equipment state is maintained to support fast operational queries without scanning large historical datasets.

By decoupling telemetry ingestion from operational workflows, the system improves scalability and reliability. Telemetry capture can continue even if downstream processing is delayed, and user-facing views can rely on a lightweight current-state store for near real-time visibility. This design also supports incremental evolution, allowing more advanced analytics and predictive maintenance features to be added without restructuring the core architecture.

5 Data and Database Design

This section explains how the platform stores, organizes, and serves data in a way that balances accuracy, scalability, and usability.
The design supports both technical needs such as telemetry ingestion and analytics, and business needs such as fast operational views, traceability, and maintainable workflows.

At a high level, the solution separates historical data from operational state. Telemetry and logs are preserved in full for analysis and audits, while a lightweight current state model ensures that dashboards and user interfaces remain fast and simple.

5.1 Data Model Overview

The data model is built around three ideas.

First, the platform uses a clear asset hierarchy so machines can always be understood in context. Every machine belongs to a production line, and every production line belongs to a plant. This hierarchy enables filtering, reporting, and access control by site and line.

Second, telemetry and events are immutable. Sensor readings and machine signals are written once and never changed. This preserves data integrity and makes the system reliable for trend analysis, root cause investigation, and compliance.

Third, the system maintains a derived current state for each machine. Instead of recalculating status from millions of telemetry records, the platform keeps one up-to-date row per machine that reflects its latest condition. This is what operational screens and supervisors rely on.

5.2 Entity Relationship Diagram

The Entity Relationship Diagram below shows the core entities such as assets, telemetry, human inputs, and maintenance workflows are connected while keeping responsibilities clearly separated. See Table 1 for field level details.

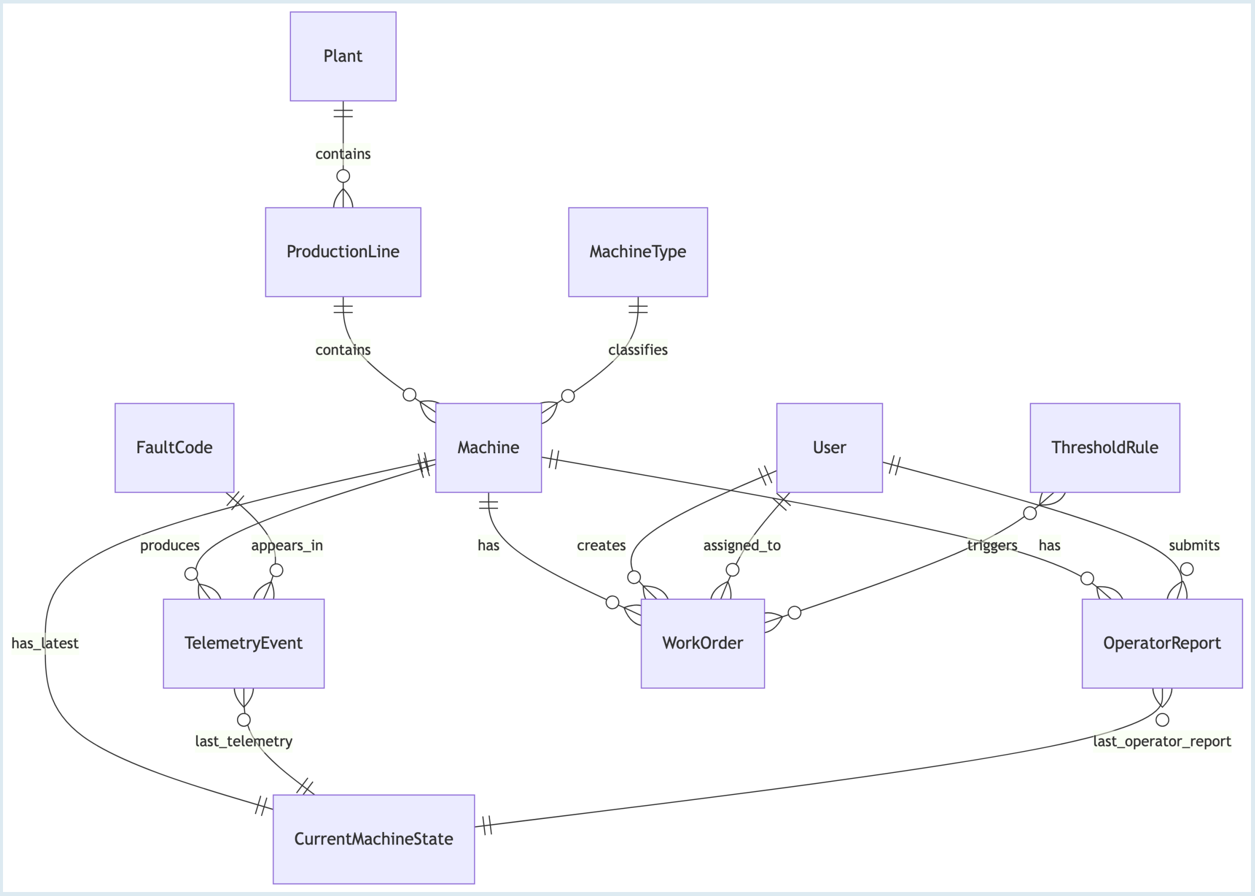

Figure 2: High level ERD for assets, telemetry, operator inputs, and maintenance

5.3 Entities and their Functions

The platform relies on a small but complete set of entities.

- Plants, production lines, machines, and machine types define the physical structure of the factory.
- Telemetry events capture raw machine signals over time.
- Operator reports capture human observations, inspections, and manual overrides.
- Work orders represent maintenance actions from creation through closure.
- Fault codes and threshold rules standardize how issues are detected and interpreted.
- Current machine state provides a fast operational snapshot for each machine.

Together, these entities allow the system to explain what happened, why it happened, and what action was taken.

5.4 Field Level Details

The table below lists the proposed tables with their primary keys, foreign keys, and essential fields required for the MVP.

Table 1: Field level details for the MVP schema

Table	Primary Key	Foreign Keys	Key Fields
Plant	plantId		plantCode, plantName, region, timezone, isActive, createdAt
ProductionLine	lineId	plantId (Plant.plantId)	lineCode, lineName, area, isActive, createdAt
MachineType	machineTypeId		typeCode, typeName, manufacturer, model, createdAt
Machine	machineId	lineId (ProductionLine.lineId), machineTypeId (MachineType.machineTypeId)	machineCode, machineName, serialNumber, installDate, isActive, createdAt
FaultCode	faultCodeId		faultCode, faultName, severity, description
TelemetryEvent	telemetryEventId	machineId (Machine.machineId), faultCodeId (FaultCode.faultCodeId)	eventTimestamp, ingestedAt, temperature, vibration, throughput, statusRaw, payloadJson
User	userId		displayName, email, role, isActive, createdAt
OperatorReport	operatorReportId	machineId (Machine.machineId), userId (User.userId)	reportTimestamp, reportType, statusOverride, comment
WorkOrder	workOrderId	machineId (Machine.machineId), createdByUserId (User.userId), assignedToUserId (User.userId)	workOrderNumber, priority, status, createdAt, closedAt
ThresholdRule	thresholdRuleId	machineTypeId (MachineType.machineTypeId)	ruleName, metricName, operator, thresholdValue, windowMinutes, severity, isActive
CurrentMachineState	machineId	machineId (Machine.machineId), lastTelemetryEventId (TelemetryEvent.telemetryEventId), lastOperatorReportId (OperatorReport.operatorReportId)	resolvedStatus, healthScore, lastUpdatedAt, openWorkOrderCount

5.5 Telemetry and Current State Strategy

Telemetry data is modeled as an immutable event stream. Each new reading is inserted and never updated. This guarantees a complete and trustworthy history that can support audits, analytics, and future use cases without redesign.

Operational views do not query this history directly. Instead, the `CurrentMachineState` table is continuously updated as telemetry arrives, operator reports are submitted, and work orders change status. Each row represents the latest resolved view of a machine, including status, health score, and maintenance indicators.

This separation between append only history and derived operational state is the key scaling decision in the design. It keeps dashboards fast and predictable while preserving full historical detail for deeper analysis and long-term insight.

6 Core Algorithms

6.1 Machine Health Scoring

The Machine Health Scoring algorithm converts raw machine signals into a single, easy-to-understand indicator that reflects how healthy a machine is at any moment. The goal is not to predict failure perfectly in the MVP, but to provide a clear, consistent signal that helps supervisors and maintenance teams prioritize attention.

Industry platforms often normalize machine health to a 0–100 scale to simplify interpretation for operations and maintenance teams I. Uptime [1]. The score combines three ideas that are common in industrial monitoring systems:

- Normal operation keeps the score high
- Abnormal sensor readings reduce the score gradually
- Faults and maintenance states reduce the score immediately and visibly

6.1.a How often the score is recalculated

The health score is recalculated each time new telemetry is processed for a machine, which in typical industrial environments means every few seconds to every minute, depending on sensor. High frequency signals such as temperature or vibration may arrive every 5–30 seconds, while lower frequency metrics such as throughput or energy usage may arrive every 1–5 minutes.

Rather than recomputing the score from all historical data, the system updates the score incrementally using the previous value. This keeps the score near real-time, computationally efficient, and stable for dashboards, while still reacting quickly when sustained issues or faults appear.

6.1.b Where the score is stored

The computed score is written to `CurrentMachineState.healthScore` for each machine:

- Table: `CurrentMachineState`
- Primary key: `machineId`
- Foreign keys:
  - `machineId -> Machine.machineId`
  - `lastTelemetryEventId -> TelemetryEvent.telemetryEventId`
  - `lastOperatorReportId -> OperatorReport.operatorReportId`
- Key fields updated by this algorithm:
  - `healthScore`
  - `lastUpdateAt`

This aligns with Table 1.

6.1.c Scoring logic overview

At each update, the score is computed using recent machine behavior, not the full telemetry history.

i Telemetry Frequency and Rolling Window

For the MVP, machines are assumed to publish telemetry approximately every 30 seconds. A 15 minute rolling window is therefore used for health scoring, providing about 30 samples per metric. This window smooths short-lived sensor noise while remaining responsive to sustained changes in machine condition.

The scoring logic combines:

- The latest telemetry readings, aggregated as rolling averages over a short time window
- Current machine status (`Running`, `Idle`, `Fault`, `UnderMaintenance`)
- Any active fault codes and their severity (`High`, `Medium`)
- Operator flags from inspections (`IssueObserved`, `MinorConcern`)

The annotated pseudocode below shows exactly how each signal contributes to the final score and how the result is stabilized for operational dashboards.

```
def compute_health_score(previous_score):
    healthScore = 100
    smoothing_param = 0.2

    # sensor penalties
    healthScore -= temperature_penalty
    healthScore -= vibration_penalty
    healthScore -= throughput_penalty

    # fault severity penalties
    if fault_severity == "High":
        healthScore -= 60
    elif fault_severity == "Medium":
        healthScore -= 30

    # apply operator input only if a report exists
    if operator_flag == "IssueObserved":
        healthScore -= 20
    elif operator_flag == "MinorConcern":
        healthScore -= 10

    # maintenance state cap
    if status == "UnderMaintenance":
        healthScore = min(healthScore, 60)

    # smoothing (EWMA)
    healthScore = (
        smoothing_param * healthScore +
        (1 - smoothing_param) * previous_score
    )
    # ensure the final score stays between 0 (worst) and 100 (best)
    healthScore = max(0, min(100, healthScore))

    return healthScore
```

Line 2
: Start from a healthy baseline score of 100 for every evaluation.

Line 3
: Define how strongly the new score reacts to recent data versus the previous score.

Lines 6-8
: Apply penalties based on sensor readings that deviate from normal operating ranges.

Line 11
: Apply a strong penalty when an active fault is detected, scaled by severity.

Line 17
: Incorporate human input from inspections or shift reports when operators flag issues.

Line 23
: Cap the score when the machine is under maintenance to reflect reduced availability.

Line 27
: Smooth the score using an Exponentially Weighted Moving Average to reduce noise.

Line 33
: Ensure the final score always remains between 0 and 100.

6.2 Latest Machine Status Resolution

The platform resolves `CurrentMachineState.resolvedStatus` using three signal sources, with clear precedence so the output stays explainable.

Precedence (highest wins):

1. UnderMaintenance when there is an open work order
2. Manual override when an operator has set a status override that is still active
3. Telemetry derived status from the latest telemetry event

This ensures an operator's safety decision can override sensors, while still allowing the system to return to automatic status when the override expires.

6.2.a Process

1. The stream processor continuously determines the latest telemetry status per machine (e.g., `Fault` if a fault code is present, else `Running` or `Idle` from the device status field).
2. The operational workflow updates `openWorkOrderCount` when work orders open or close.

3. The latest operator report with `statusOverride` (if any) is treated as a manual override for a short time window (for example 2 to 8 hours), after which the system falls back to telemetry.

6.2.b Query

```
WITH LatestTelemetry AS (
  SELECT
    t.machineId,
    t.telemetryEventId,
    t.eventTimestamp,
    t.statusRaw,
    t.faultCodeId,
    ROW_NUMBER() OVER (
      PARTITION BY t.machineId
      ORDER BY t.eventTimestamp DESC
    ) AS rn
  FROM TelemetryEvent t
),

LatestOperatorOverride AS (
  SELECT
    r.machineId,
    r.operatorReportId,
    r.reportTimestamp,
    r.statusOverride,
    ROW_NUMBER() OVER (
      PARTITION BY r.machineId
      ORDER BY r.reportTimestamp DESC
    ) AS rn
  FROM OperatorReport r
  WHERE r.statusOverride IS NOT NULL
),

OpenWorkOrders AS (
  SELECT
    w.machineId,
    COUNT(*) AS openWorkOrderCount
  FROM WorkOrder w
  WHERE w.status IN ('Open', 'InProgress')
  GROUP BY w.machineId
)

SELECT
  m.machineId,
  CASE
    WHEN COALESCE(owo.openWorkOrderCount, 0) > 0 THEN 'UnderMaintenance'

    WHEN loo.statusOverride IS NOT NULL
```

```
     AND loo.reportTimestamp >= CURRENT_TIMESTAMP - INTERVAL '4 hours' THEN
loo.statusOverride

    WHEN lt.faultCodeId IS NOT NULL THEN 'Fault'
    ELSE COALESCE(lt.statusRaw, 'Idle')
  END AS resolvedStatus,

  COALESCE(owo.openWorkOrderCount, 0) AS openWorkOrderCount,
  lt.telemetryEventId AS lastTelemetryEventId,
  loo.operatorReportId AS lastOperatorReportId,
  GREATEST(
    COALESCE(lt.eventTimestamp, TIMESTAMP '1970-01-01'),
    COALESCE(loo.reportTimestamp, TIMESTAMP '1970-01-01')
  ) AS lastUpdateAt

FROM Machine m
LEFT JOIN LatestTelemetry lt
  ON m.machineId = lt.machineId AND lt.rn = 1
LEFT JOIN LatestOperatorOverride loo
  ON m.machineId = loo.machineId AND loo.rn = 1
LEFT JOIN OpenWorkOrders owo
  ON m.machineId = owo.machineId;
```

Line 1
: Pick the most recent telemetry event per machine.

Line 15
: Pick the most recent operator report that contains a manual status override.

Line 29
: Count open work orders per machine to detect maintenance state.

Line 40
: Resolve status using precedence: maintenance → manual override (time boxed) → telemetry.

Line 51
: Keep pointers to the exact records that drove the current view for auditability.

Line 53
: lastUpdateAt reflects the most recent update source (telemetry vs operator override).

Line 60
: Only one telemetry row per machine is used (rn = 1) to avoid scanning history in dashboards.

7 Dashboards and User Experience

7.1 Current Equipment Status Screen

7.2 Operational and Maintenance Views

8 Security and Data Protection

9 Scalability and Reliability

9.1 Data Growth and Performance

9.2 Failure Scenarios and Resilience

10 Conclusion and Recommendations

10.1 Summary

10.2 Future Enhancements

11 References

Bibliography

[1] I. Uptime, “Understanding Machine Health Score and Its Importance in Asset Reliability.” Accessed: Dec. 15, 2022. [Online]. Available: https://medium.com/@infiniteuptime/understanding-machine-health-score-and-its-importance-in-asset-reliability-4912218ea2f7