Factory Machine Event Backend System

Complete Technical Documentation

Technology Stack

Language: Java

Framework: Spring Boot

Database: PostgreSQL

Architecture: Layered MVC Pattern

Backend Intern Assignment

Complete System Design & Implementation Guide

January 2026 - Sujeet Prajapati

1. System Overview

This backend system manages events from factory machines, providing real-time ingestion, validation, deduplication, and statistical analysis capabilities. The system is designed to handle high-throughput concurrent requests while maintaining data integrity and performance.

Key Objectives

- **Receive and Store:** Accept batch machine events with validation
- **Process Efficiently:** Handle 1000+ events per second
- **Ensure Accuracy:** Implement deduplication and update logic
- **Provide Analytics:** Generate machine statistics and defect reports

System Architecture

The system follows a clean layered architecture with clear separation of concerns:

Layer 1: Controller - REST API endpoints for client interaction

Layer 2: Service - Business logic, validation, and processing

Layer 3: Repository - Data access and persistence operations

Layer 4: Database - PostgreSQL with optimized schema and indexes

Request Flow:
Client → Controller → Service → Repository → Database

Core Components

Component	Responsibility
EventIngestionController	Handles batch event ingestion requests
StatsController	Provides analytics and statistics endpoints
EventIngestionService	Core processing engine with validation logic
StatsService	Computes machine statistics and rankings
MachineEventRepository	Database operations and query execution

2. Event Structure & Validation

Event JSON Schema

```
{ "eventId": "E-123", "eventTime": "2026-01-15T10:12:03.123Z", "receivedTime":
"2026-01-15T10:12:04.500Z", "machineId": "M-001", "lineId": "LINE-A", "factoryId":
"F01", "durationMs": 4312, "defectCount": 0 }
```

Field Descriptions

Field	Type	Description
eventId	String	Unique identifier for deduplication
eventTime	Instant	When event occurred (used for queries)
receivedTime	Instant	When backend received event (auto-set)
machineId	String	Machine identifier
durationMs	Long	Event duration in milliseconds
defectCount	Integer	Number of defects (-1 = unknown)

Validation Rules

Critical Validation Checks

Rule 1: Duration Validation

```
REJECT if: durationMs < 0 OR durationMs > 21,600,000 (6 hours = 6 × 60 ×
60 × 1000 = 21,600,000 milliseconds)
```

Rule 2: Future Time Validation

```
maxAllowedTime = currentTime + 15 minutes REJECT if: eventTime >
maxAllowedTime
```

Rule 3: Defect Count Validation

```
REJECT if: defectCount < -1
```

Special Handling Rules

receivedTime Override

The system automatically sets receivedTime to current timestamp. Client value is ignored.

```
receivedTime = Instant.now() // System-set
```

defectCount = -1 Handling

When defectCount = -1, it indicates "unknown" defects:

- ✓ Event is stored in database
- ✓ Counted in total events
- ✗ Excluded from defect calculations

3. Deduplication & Update Logic

The Challenge

Factory machines may send duplicate events due to network retries or system restarts. The system must intelligently handle these scenarios.

Payload Hash Generation

```
hashInput = eventTime + machineId + lineId + factoryId + durationMs +
defectCount payloadHash = SHA256(hashInput)
```

```
String hashInput = event.getEventTime() + event.getMachineId() + event.getLineId() +
event.getFactoryId() + event.getDurationMs() + event.getDefectCount(); String
payloadHash = DigestUtils.sha256Hex(hashInput);
```

Decision Logic Flow

Step-by-Step Processing

Step 1: Query database by eventId

Step 2: Check if event exists

- If NOT EXISTS → Insert new event
- If EXISTS → Proceed to Step 3

Step 3: Compare payload hashes

- If SAME HASH → Duplicate, ignore (dedupe count++)
- If DIFFERENT HASH → Proceed to Step 4

Step 4: Compare receivedTime

- If NEWER receivedTime → Update (update count++)
- If OLDER receivedTime → Ignore outdated data

Decision Matrix

Scenario	eventId	Hash	Time	Action
New Event	Not Found	-	-	INSERT
Exact Duplicate	Found	Same	-	DEDUPE
Updated Event	Found	Different	Newer	UPDATE
Outdated	Found	Different	Older	IGNORE

Update Logic Formula

```
shouldUpdate = (existingEvent.eventId == newEvent.eventId) AND
(existingEvent.payloadHash != newEvent.payloadHash) AND
(newEvent.receivedTime > existingEvent.receivedTime)
```

4. API Endpoints

Endpoint 1: Batch Ingestion

```
POST /api/events/batch Content-Type: application/json Request: Array of events
[{"eventId":"E-1", "eventTime":"...", ...}]
```

Response Structure

```
{ "accepted": 950, "deduped": 30, "updated": 10, "rejected": 10, "rejections":
[ {"eventId":"E-99","reason":"INVALID_DURATION"} ] }
```

Endpoint 2: Machine Statistics

```
GET /api/stats?machineId=M-001 &start=2026-01-15T00:00:00Z &end=2026-01-15T06:00:00Z
```

```
Response: { "machineId": "M-001", "eventsCount": 1200, "defectsCount": 13,
"avgDefectRate": 2.17, "status": "Warning" }
```

Calculation Formulas: eventsCount = COUNT(all valid events in window) defectsCount = SUM(defectCount) WHERE defectCount != -1 windowHours = (end - start) in seconds / 3600.0 avgDefectRate = defectsCount / windowHours status = IF(avgDefectRate < 2.0) "Healthy" ELSE "Warning"

Time Window Rules

- **start:** Inclusive (included)
- **end:** Exclusive (not included)
- **Uses:** eventTime (not receivedTime)

Endpoint 3: Top Defect Lines

```
GET /api/stats/top-defect-lines ?factoryId=F01 &from=2026-01-15T00:00:00Z &to=2026-
01-15T23:59:59Z &limit=10
```

Calculation per Line:

```
totalDefects = SUM(defectCount) WHERE defectCount != -1 eventCount = COUNT(events) defectsPercent = (totalDefects / eventCount) × 100 ORDER BY totalDefects DESC LIMIT n
```

5. Database Schema

MachineEventEntity Table

Column	Type	Constraints
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT
eventId	VARCHAR(255)	UNIQUE, NOT NULL
eventTime	TIMESTAMP	NOT NULL
receivedTime	TIMESTAMP	NOT NULL
machineId	VARCHAR(100)	NOT NULL
lineId	VARCHAR(100)	NULLABLE
factoryId	VARCHAR(100)	NULLABLE
durationMs	BIGINT	NOT NULL
defectCount	INTEGER	NOT NULL
payloadHash	VARCHAR(64)	NOT NULL

Database Indexes

Performance Optimization Indexes

Index 1: Unique Event ID

```
CREATE UNIQUE INDEX idx_event_id ON machine_event(eventId);
```

Index 2: Machine Statistics Query

```
CREATE INDEX idx_machine_time ON machine_event(machineId, eventTime);
```

Index 3: Factory Line Analytics

```
CREATE INDEX idx_factory_line_time ON machine_event(factoryId, lineId,
eventTime);
```

Why These Indexes?

- **idx_event_id:** Fast deduplication lookup O(1)
- **idx_machine_time:** Efficient stats queries by machine
- **idx_factory_line_time:** Optimized top-defect-lines queries

6. Thread Safety & Performance

Thread Safety Mechanisms

Layer 1: Transactional Boundaries

```
@Transactional public BatchIngestionResponse ingestBatch( List events) { // All
operations in single transaction // Either all succeed or all rollback }
```

Layer 2: Database Constraints

- **UNIQUE constraint** on eventId prevents duplicates
- **Row-level locks** prevent concurrent modifications
- **Isolation level** ensures consistent reads

```
@Lock(LockModeType.PESSIMISTIC_WRITE) Optional findByEventIdForUpdate(String
eventId);
```

Layer 3: Atomic Operations

- Single INSERT/UPDATE per event
- No intermediate states visible
- ACID properties maintained

Concurrency Test Scenario

```
// 20 parallel threads sending batches ExecutorService executor =
Executors.newFixedThreadPool(20); for (int i = 0; i < 20; i++) { executor.submit(()
-> { client.post("/api/events/batch", generateBatch(100)); }); } // Verify: No
corruption, correct counts
```

Performance Strategies

Strategy	Implementation	Impact
Batch Processing	Single transaction	1000x less DB trips
Hash Comparison	SHA-256 hash	O(1) vs O(n)
DB Indexes	Strategic indexes	O(log n) queries
Bulk Operations	JPA batch insert	10x faster

Performance Target: Process 1000 events in < 1 second Target throughput = 1000 events/sec Average time per event < 1ms

7. Testing Strategy (8 Mandatory Tests)

Test 1: Exact Duplicate Detection

Scenario: Same eventId, identical payload

Expected: First accepted, second deduped

```
Event e1 = createEvent("E-1", "M-001", 1000, 0); Event e2 = createEvent("E-1",
"M-001", 1000, 0); Response r = ingestBatch(Arrays.asList(e1, e2));
assertEquals(1, r.getAccepted()); assertEquals(1, r.getDeduped());
```

Test 2: Update with Newer receivedTime

Expected: Second updates first

```
Event e1 = createEvent("E-2", "M-001", 1000, 0);
ingestBatch(Arrays.asList(e1)); Thread.sleep(100); Event e2 = createEvent("E-
2", "M-001", 1500, 2); Response r = ingestBatch(Arrays.asList(e2));
assertEquals(1, r.getUpdated());
```

Test 3: Ignore Older receivedTime

```
Event newer = createEventWithTime("E-3", now());
ingestBatch(Arrays.asList(newer)); Event older = createEventWithTime("E-3",
now().minus(10, MINUTES)); Response r = ingestBatch(Arrays.asList(older));
assertEquals(0, r.getUpdated());
```

Test 4: Invalid Duration Rejected

```
Event negative = createEvent("E-4", -100); Event tooLong = createEvent("E-5",
22000000); Response r = ingestBatch(List.of(negative, tooLong));
assertEquals(2, r.getRejected());
```

Test 5: Future Event Rejected

```
Instant future = now().plus(20, MINUTES); Event e = createEventWithTime("E-6",
future); Response r = ingestBatch(Arrays.asList(e)); assertEquals(1,
r.getRejected());
```

Test 6: defectCount=-1 Excluded

```
Event e1 = createEvent("E-7", 1000, 5); Event e2 = createEvent("E-8", 1000,
-1); Event e3 = createEvent("E-9", 1000, 3); ingestBatch(List.of(e1, e2, e3));
Stats s = getStats("M-001", start, end); assertEquals(3, s.getEventsCount());
assertEquals(8, s.getDefectsCount());
```

Test 7: Time Boundary Correctness

```
Instant start = parse("2026-01-15T10:00:00Z"); Instant end = parse("2026-01-
15T11:00:00Z"); Event atStart = createTime("E-10", start); Event middle =
createTime("E-11", start+1800s); Event atEnd = createTime("E-12", end);
ingestBatch(List.of(atStart, middle, atEnd)); Stats s = getStats("M-001",
start, end); assertEquals(2, s.getEventsCount());
```

Rule: start INCLUSIVE, end EXCLUSIVE

Test 8: Thread-Safety Concurrent Ingestion

```
ExecutorService exec = Executors.newFixedThreadPool(20); for (int i = 0; i <
20; i++) { exec.submit(() -> { ingestBatch(generateBatch(50)); }); } // Verify
no data corruption long total = repository.count(); assertTrue(total > 0);
assertEquals(expectedCount, total);
```

Setup & Run Instructions

Prerequisites

```
✓ Java 17 or higher ✓ Maven 3.6+ ✓ PostgreSQL 12+ ✓ IDE (IntelliJ/Eclipse)
```

Database Setup

```
CREATE DATABASE factorydb; CREATE USER factoryuser WITH PASSWORD 'pass'; GRANT ALL
PRIVILEGES ON DATABASE factorydb TO factoryuser;
```

Application Configuration

Edit `application.properties`:

```
spring.datasource.url= jdbc:postgresql://localhost:5432/factorydb
spring.datasource.username=postgres spring.datasource.password=postgres
spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true
```

Build & Run

```
mvn clean install mvn test mvn spring-boot:run
```

8. Performance Benchmark Results

System Specifications

Component	Specification
Processor	Intel Core i7-10750H @ 2.60GHz
RAM	16 GB DDR4
Storage	512 GB NVMe SSD
Database	PostgreSQL 14.5
Java	OpenJDK 17.0.2

Benchmark Results

Test: 1000 Events (New Inserts)

```
Command: mvn test -Dtest=BenchmarkTest Batch Size: 1000 events Execution Time:
847ms Throughput: 1,180 events/second Status: ✓ PASS (< 1 second)
```

Test: 1000 Events (30% Duplicates)

```
Execution Time: 923ms Dedup Overhead: 76ms Status: ✓ PASS
```

Test: 20 Concurrent Threads

```
Total Events: 20,000 Total Time: 4.2 seconds Throughput: 4,761 events/second
Status: ✓ PASS
```

Future Improvements

Performance Enhancements

- Redis caching for stats queries
- Async processing queue
- Database partitioning by time
- Read replicas for analytics

New Features

- Add max/min duration metrics
- Percentile calculations (P95, P99)
- Real-time monitoring dashboard
- Alert system for thresholds

Operational Improvements

- Swagger/OpenAPI documentation
- Docker containerization
- CI/CD pipeline
- Enhanced logging & metrics

9. Quick Reference Guide

Key Formulas Summary

Validation Formulas: durationMs: 0 ≤ value ≤ 21,600,000 eventTime: value ≤ currentTime + 15 minutes defectCount: value ≥ -1

Payload Hash: hashInput = eventTime + machineId + lineId + factoryId + durationMs + defectCount payloadHash = SHA256(hashInput)

Deduplication Logic: IF eventId NOT EXISTS → INSERT IF eventId EXISTS AND hash SAME → DEDUPE IF eventId EXISTS AND hash DIFF AND time NEWER → UPDATE IF eventId EXISTS AND hash DIFF AND time OLDER → IGNORE

Machine Statistics: eventsCount = COUNT(all valid events) defectsCount = SUM(defectCount) WHERE defectCount ≠ -1 windowHours = (endTime - startTime) in seconds / 3600.0 avgDefectRate = defectsCount / windowHours status = IF avgDefectRate < 2.0 THEN "Healthy" ELSE "Warning"

Top Defect Lines: totalDefects = SUM(defectCount) WHERE defectCount ≠ -1 eventCount = COUNT(events) defectsPercent = ROUND((totalDefects / eventCount) × 100, 2) ORDER BY totalDefects DESC LIMIT n

Edge Cases Handled

Edge Case	Solution
Empty batch	Return success with 0 counts
Partial failure	Process valid, return rejections
Clock skew	Allow 15-min future tolerance
Concurrent same ID	DB constraint prevents dupes
Missing fields	Store as NULL, handle in queries

API Testing Examples

```
# Batch Ingestion curl -X POST http://localhost:8080/api/events/batch \ -H "Content-
Type: application/json" \ -d '[{"eventId":"E-1",...}]' # Get Statistics curl
"http://localhost:8080/api/stats?machineId=M-001 &start=2026-01-15T00:00:00Z
&end=2026-01-15T23:59:59Z" # Top Defect Lines curl
"http://localhost:8080/api/stats/top-defect-lines ?factoryId=F01&limit=10"
```

End of Documentation

Factory Machine Event Backend System
Complete Technical Reference - January 2026