Backend Intern Assignment

Tech stack

- **Language:** Java
- **Framework:** Spring or Play
- You are free to use any database, but your project must run locally with clear instructions.

Using external help / LLMs

We want you to do well in the assignment, and you are free to take inspiration or help from anywhere (including LLMs, blogs, articles, videos, etc.).
However, please keep in mind the following

- You must **deeply understand each and every line of code you submit**.
- In the interview, we may ask you to:
    - Explain why something is implemented a certain way
    - Trace a bug from tests
    - Make changes **in real time** (screen-share + edit + run)
    - Add a test for a new edge case

If you cannot explain your own code, we will not be able to proceed with the candidature.

What you're building

Imagine a factory with many machines running all day. Each machine does the following apart from a lot of non-relevant things for our usecase:

- Produces items
- Sometimes produces defective items
- Sends small messages ("events") to a backend system whenever something happens

Your job is to build that backend system. This backend system has the following set of responsibilities:

- Receive and store machine events

- Answer questions (stats) about what happened during a specific time period

---

Event format

Each event is JSON:

```
{
  "eventId": "E-123",
  "eventTime": "2026-01-15T10:12:03.123Z",
  "receivedTime": "2026-01-15T10:12:04.500Z",
  "machineId": "M-001",
  "durationMs": 4312,
  "defectCount": 0
// What else
}
```

Rules / assumptions

- Use `eventTime` for query windows (not receivedTime).
- `receivedTime` is only for reference in the json, but would be set by our service and data if any in the API request payload would be ignored.
- **Deduplication by `eventId`:**
  - same `eventId` + identical payload → dedupe (ignore)
  - same `eventId` + different payload → treat as an **update**
- Validation:
  - reject if `durationMs < 0` or `durationMs > 6 hours`
  - reject if `eventTime` is > 15 minutes in the future
- **Special defect rule:** `defectCount = -1` means “unknown” → store event but **ignore it for defect calculations**.

Required Endpoints

1. Ingest (Batch) | `POST /events/batch`

Input: A JSON array containing **100s of events**.

Example request body:

```
[
  { "eventId":"E-1", "eventTime":"...", "receivedTime":"...",
"machineId":"M-001", "durationMs":1000, "defectCount":0 },
  { "eventId":"E-2", "eventTime":"...", "receivedTime":"...",
"machineId":"M-001", "durationMs":1500, "defectCount":-1 }
]
```

Response (suggested; you can add fields):

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

Requirements

- Must be **thread-safe**. Assume multiple sensors push concurrently.
    - You should simulate concurrency in tests (e.g., 5–20 parallel requests).
- **Performance requirement:** must process a batch of **1,000 events in under 1 second** on a standard laptop.
    - Include benchmark instructions + your measured result in `BENCHMARK.md`.

B) Query Stats | `GET /stats?machineId=M-001&start=...&end=...`

- `start` inclusive, `end` exclusive

Returns:

- `eventsCount`: total **valid** events in the window (post-validation, after applying dedupe/update)
- `defectsCount`: total defects for the window, **ignoring defectCount = -1**

- `avgDefectRate:` average defects/hour across the window
    - calculation: `defectsCount / windowHours` (windowHours = duration in seconds / 3600.0)
- `status`:
    - `"Healthy"` if avg defect rate `< 2.0`
    - else `"Warning"`

Example response:

```
{
  "machineId": "M-001",
  "start": "2026-01-15T00:00:00Z",
  "end": "2026-01-15T06:00:00Z",
  "eventsCount": 1200,
  "defectsCount": 6,
  "avgDefectRate": 2.1,
  "status": "Warning"
}
```

C) Top defect lines endpoint | `GET /stats/top-defect-lines?factoryId=F01&from=...&to=...&limit=10`

Return list of:

- `lineId`
- `totalDefects`
- `eventCount`
- `defectsPercent` (Defect per 100 Events, rounded to 2 decimals)

Tests (mandatory)

Minimum **8 tests**, covering at least:

1. Identical duplicate eventId → deduped
2. Different payload + newer receivedTime → update happens

3. Different payload + older receivedTime → ignored
4. Invalid duration rejected
5. Future eventTime rejected
6. DefectCount = -1 ignored in defect totals
7. start/end boundary correctness (inclusive/exclusive)
8. Thread-safety test: concurrent ingestion doesn't corrupt counts / create incorrect updates

---

README / Documentation requirements (mandatory)

Your `README.md` must explain the approach **in depth**. At minimum include:

1. **Architecture**
2. **Dedupe/update logic:** how you compare payloads, how you decide "winning" record
3. **Thread-safety:** what makes it thread-safe (locks? DB constraints? transactional semantics? etc.)
4. **Data model:** table schema or in-memory structures
5. **Performance strategy:** what you did to process 1000 events within 1 sec
6. **Edge cases & assumptions:** what you chose and why, tradeoffs if any
7. **Setup & run instructions**
8. **What you would improve with more time**

Benchmark (mandatory)

Create `BENCHMARK.md` and include:

- Your laptop/desktop specs (CPU/RAM)
- Command used to run the ingestion benchmark
- Measured timing for:
  - ingesting one batch of **1000 events**
- Any optimizations you attempted

Interview follow-up (important)

In the interview, we may ask you to:

- Add a new metric (e.g., `max_duration_ms`)
- Add a new filter (e.g., only include events where durationMs > X)
- Change health threshold rules

- Fix a new edge case we introduce

So please submit something you truly understand.