Technical Test
Fullstack Developer

Mini Attendance System

Tech Stack :

- Backend: **Node.js** (**Typescript**) atau **Go**
- Frontend **: Bebas (React.js, Vue.js, Svelte, Angular, dsb.)**
- DB: **MySQL** / **PostgreSQL**
- Broker: **Kafka** / **RabbitMQ**
- Cache: **Redis**
- Container: **Docker**

Petunjuk :

1. **Implementasikan mekanisme Auth Token (JWT)** untuk memberikan akses ke API secara aman.
2. **Bangun fungsi Check-in dan Check-ou**t untuk mencatat kehadiran karyawan.
3. **Sediakan fitur laporan absensi** yang dapat menampilkan, memfilter, dan mencetak data kehadiran dengan status: hadir, terlambat, pulang cepat, atau tidak hadir.
4. **Gunakan pendekatan event-driven** pada proses *check-in* dan *check-out*:
   - **Service utama** hanya mencatat event ke database dan menerbitkannya ke **Message Broker** (Kafka atau RabbitMQ).
   - **Service Report (Consumer)** membaca event dari broker untuk membuat atau memperbarui data laporan absensi secara asinkron di database.
5. **Buat file** `README.md` yang berisi petunjuk instalasi, konfigurasi environment, dan langkah menjalankan project (termasuk **`docker compose up`**).
6. **Sertakan dokumentasi API** menggunakan Swagger atau Postman Collection.
7. **Gunakan GitHub atau GitLab** untuk mengelola kode dan push project.
8. **Gunakan Docker** untuk melakukan kontainerisasi seluruh service (Auth, Attendance, Report, Broker, Database, Redis).
9. **Tambahkan observability dasar** berupa *structured logging* dan minimal satu *health check endpoint* untuk memantau kondisi keseluruhan sistem dari satu endpoint.
10. Kandidat diperbolehkan menggunakan library tambahan untuk logging, broker client, atau ORM sesuai preferensi, selama sesuai dengan prinsip arsitektur yang diminta.

Aturan Bisnis :

- Tidak boleh double check-in sebelum check-out.

Pengumpulan :

Deadline: 09 November 2025 23:59 WIB
Kirim link repo publik ke `hrd@ts.co.id` , cc: `satriyopnr@ts.co.id`

Technical Test
Fullstack Developer

Mini Attendance System

This technical test is designed to evaluate your ability to design and implement a small but realistic distributed system.
You will build a Mini Attendance System that records employee check-in and check-out activities, generates reports asynchronously, and demonstrates your understanding of clean architecture, asynchronous processing, and service decoupling.

Tech Stack :

- Backend: Node.js (Typescript) or Go
- Frontend : React / Vue
- DB: MySQL / PostgreSQL
- Cache: Redis
- Container: Docker

Instructions :

1. Implement a token-based authentication mechanism for secure API access.
2. Build check-in and check-out features to record employee attendance.
3. Provide an attendance report feature that can display, filter, and export data with statuses such as *present*, *late*, *early leave*, or *absent*.
4. Apply an asynchronous event propagation approach for the check-in and check-out process:
    a. The main service records attendance events to the database and emits them asynchronously.
    b. A report service consumes these events and updates summaries in the background. Reports should not block user requests.
    c. The delivery mechanism (message queue, stream, or background worker) is up to you, as long as the flow is asynchronous and decoupled.
5. Include a README.md containing installation steps, environment setup, and instructions to run the project (including `docker compose up`).
6. Add API documentation using Swagger or a Postman Collection.
7. Use GitHub or GitLab for source-code management and push the project there.
8. Use Docker to containerize all components (Auth, Attendance, Report, Database, Redis, and the event propagation layer).
9. Implement basic observability: structured logging and at least one health-check endpoint to monitor overall system status.
10. You may use any additional libraries for logging, async processing, etc as long as they align with the architectural requirements.

Business Rules :

1. Double check-in before check-out is not allowed.

Meeting Room Scheduler

Objective

Given a list of meetings with start and end times, determine the maximum number of meetings that can be scheduled without overlapping in a single room.

Requirements

Input:
An array of objects, each representing a meeting with `start` and `end` times (in 24-hour format, `"HH:MM"`).

```
 [
   { "id": 1, "start": "09:00", "end": "10:30" },
   { "id": 2, "start": "09:45", "end": "11:00" },
   { "id": 3, "start": "10:40", "end": "12:00" },
   { "id": 4, "start": "13:00", "end": "14:00" }
]
```

Output:
A list of selected meeting IDs that can fit in the schedule without overlap, maximizing the total number of meetings.

```
 {
   "scheduled": [1, 3, 4],
   "count": 3
}
```

Rules:

1. Meetings cannot overlap.
2. You may assume all inputs are on the same day.

Submission :

Deadline: 09 November 2025 23:59 WIB

Send public repository link to `hrd@ts.co.id` , cc: `satriyopnr@ts.co.id`