**Low-Level Design (LLD)**

**1. Introduction**

This LLD provides implementation-level details for integrating attendance devices (biometric, RFID, QR, face recognition) with an LMS backend using **Python** as the core programming language and a **GUI-based administration tool** (Tkinter or PyQt) for local management.

**2. Technology Stack**

* **Language:** Python 3.11+
* **Backend Framework:** Flask / FastAPI (REST APIs)
* **GUI Framework:** Tkinter or PyQt5 (for admin console)
* **Database:** PostgreSQL
* **Messaging:** Kafka (via confluent-kafka-python)
* **Deployment:** Docker Compose / Kubernetes
* **Device Communication:** REST API, WebSocket, or file-based (CSV/JSON)

**3. Component Overview**

1. **Python Device Integration Service** — Collects attendance logs from devices and normalizes data.
2. **GUI Admin Tool** — Local application for managing devices, viewing logs, triggering manual syncs.
3. **Attendance Processing Service** — Flask/FastAPI app that validates and stores attendance.
4. **Kafka Topics** — Decouples ingestion and processing.
5. **PostgreSQL Database** — Stores normalized attendance data.
6. **Notification Service** — Python-based alerts (SMTP, Twilio).

**4. GUI Design**

**Main Screens:**

1. **Dashboard:** Device status, last sync time, total attendance today.
2. **Device Management:** Add/Edit/Delete devices, configure API keys.
3. **Attendance Logs:** Filter by date, device, user.
4. **Manual Sync:** Trigger on-demand device fetch.
5. **Settings:** Kafka brokers, DB connection, retry policies.

**Example Tkinter Layout:**

from tkinter import \*

root = Tk()

root.title("LMS Attendance Admin Console")

Label(root, text="Device Status").grid(row=0, column=0)

Button(root, text="Sync Now").grid(row=0, column=1)

root.mainloop()

**5. Canonical Data Model (Python Dict)**

attendance\_event = {

"event\_id": "uuid",

"device\_id": "string",

"device\_type": "BIOMETRIC|RFID|QR|FACE",

"capture\_type": "IN|OUT|SWIPE",

"timestamp": "2025-08-11T09:11:00Z",

"user\_identifier": {

"type": "EMP\_ID|CARD\_ID|BIO\_ID|FACE\_HASH",

"value": "string"

},

"location": {

"site\_id": "string",

"geo": {"lat": 0.0, "lon": 0.0}

},

"raw\_payload": {},

"metadata": {"firmware\_version": "1.0"}

}

**6. APIs**

* Flask/FastAPI endpoints for device push (/api/v1/devices/events) and GUI-triggered pulls (/api/v1/devices/{id}/fetch).
* REST endpoint /api/v1/attendance/events for storing processed attendance in LMS.

**7. Kafka Messaging**

* attendance.raw — raw device events
* attendance.normalized — enriched validated events

Python producer example:

from confluent\_kafka import Producer

p = Producer({'bootstrap.servers': 'kafka:9092'})

p.produce('attendance.raw', key='event\_id', value=json.dumps(attendance\_event))

**8. Database Schema**

(Same as previous LLD but implemented with Python’s psycopg2 or SQLAlchemy ORM.)

**9. Error Handling**

* GUI shows failed sync attempts with retry button.
* Retry mechanism with exponential backoff in Python.

**10. Security**

* JWT token auth for API requests.
* Local encryption for sensitive GUI config values.

**11. Monitoring**

* GUI shows real-time Kafka lag & device health.
* Python logging module for structured logs.

**12. Deployment**

* Python services in Docker.
* GUI packaged with PyInstaller for Windows/Linux.

**13. Testing Strategy**

* Pytest for unit/integration tests.
* GUI automated tests with pytest-qt (if PyQt) or unittest + pyautogui (if Tkinter).