BP-G1-FOLR-Eng
FOLR Engine Processor (Face-Only Login & Registration – Engine Stack)

1. Scope

This engine listens for Kafka events triggered by the Entry controller for either face registration or login. It:

- Reads the Redis key for session metadata
- Fetches the face image file (AES-256 encrypted) stored in PostgreSQL
- Decrypts the file in memory
- Converts the image into a 512-d face vector using ONNX
- Matches the vector against FAISS face index (existing registered users)
- Classifies outcome as:
    - new – face not found
    - old – existing face
    - match – login successful
    - no_match – login failed
    - flagged – suspicious duplicate face
- Updates Redis key with result, vector, and session flags
- Emits Kafka event on task completion

This service is **non-REST**, designed to run as a pure background microservice.

2. Objective

🎯 **Primary Objective:**

To implement a **robust face recognition engine** that ensures:

- Full in-memory face vectorization pipeline
- Accurate match detection using FAISS similarity
- Consistent session result updates in Redis
- Asynchronous Kafka processing with traceable event logs

🔍 **Component-wise Breakdown:**

✅ **Step 1 – Kafka Consumer**

- Topic: FOLR_REGISTER_OR_LOGIN
- Payload includes:
  - session_id, email, mac_id, type, image_url

✅ **Step 2 – PostgreSQL Fetch**

- Retrieve encrypted face image (blob) from DB
- Table: folr_image_blob
- Use session_id to query

✅ **Step 3 – AES-256 Decryption**

- CBC Mode Decryption
- IV stored with encrypted blob
- Key from .env or config

✅ **Step 4 – Face Image Preprocessing**

- Use OpenCV to convert raw bytes → RGB image
- Normalize, resize (e.g., 112x112 or 224x224)
- Validate image dimensions and content

✅ **Step 5 – ONNX Vectorization**

- Load ONNX model for face embedding
- Input: preprocessed image
- Output: 512-d float vector (Face Embedding)

✅ Step 6 – FAISS Search

- Depending on type:
  - **register**:
    - Check similarity with existing vectors
    - If match found → old or flagged
    - If no close match → new
  - **login**:
    - Match with user's registered vector
    - If similarity > threshold → match
    - Else → no_match

✅ Step 7 – Redis Update

- Key: FOLR_SESSION:<session_id>
- Write:

```
{
  "status": "COMPLETED",
  "result": "new",
  "vector": [0.012, 0.893, ..., 0.110],
  "score": 0.85
}
```

✅ Step 8 – Kafka Emit

- Topic: TASK_COMPLETED_FOLR
- Notifies Entry controller of task completion

3. Tech Stack and Tools

Layer	Tools/Libs Used
Kafka	confluent-kafka-python

Layer	Tools/Libs Used
Redis	redis-py, session JSON keys
PostgreSQL	psycopg2 for blob and metadata
AES Decryption	cryptography (AES-256 CBC mode)
Face Processing	opencv-python, numpy
ONNX Runtime	onnxruntime, face model
FAISS Engine	faiss-cpu, in-memory vector index
Logging	Custom session-aware loggers

4. Kafka and Redis Overview

Kafka Topics

Direction	Topic Name	Description
Consume	FOLR_REGISTER_OR_LOGIN	Entry-triggered task
Produce	TASK_COMPLETED_FOLR	Engine task completion

Redis Keys

Redis Key	TTL	Fields
FOLR_SESSION:<sid>	180s	result, vector, score, status

Redis is updated after ONNX and FAISS flow completes, ensuring atomic result handling per session.

5. REST API Specification

✕ Not Applicable — This service is Kafka and Redis-only.

6. Kafka Message Schema

Consumed ← FOLR_REGISTER_OR_LOGIN

{
"session_id": "abc-456",
"email": "person@tmachine.ai",
"type": "register",
"image_url": "/blob/image/abc-456"
}

Produced → TASK_COMPLETED_FOLR

{
"session_id": "abc-456",
"status": "COMPLETED",
"result": "new",
"score": 0.87
}

7. Redis Key Strategy

Redis Field Purpose

result	match, no_match, new, old, flagged
vector	512-d float array from ONNX
score	Similarity score from FAISS
status	PENDING, COMPLETED, ERROR

Keys are created by Entry stack and updated by Engine upon processing.

8. Suggested Folder Structure

```
bp_g1_folr_engine/
├── kafka/
│   ├── consumer.py
│   └── producer.py
```

```
├── redis/
│   ├── session_manager.py
├── db/
│   ├── blob_handler.py
│   └── connection.py
├── crypto/
│   └── aes_cbc_decryptor.py
├── image/
│   └── preprocessor.py
├── onnx/
│   └── face_vectorizer.py
├── faiss/
│   └── search_and_classify.py
├── utils/
│   └── logger.py
└── config/
    └── env.py
```

9. Project Completion Checklist

☑	Task	Description
✔	Kafka Listener Setup	Handles FOLR_REGISTER_OR_LOGIN messages
✔	PostgreSQL Blob Retrieval	Fetches encrypted face image
✔	AES Decryption	AES-256 CBC decryption in memory
✔	Image Preprocessing	Resize, validate, normalize

☑ Task	Description
✔ ONNX Inference	Extracts face embeddings (512-d)
✔ FAISS Search	Matches face vectors efficiently
✔ Result Categorization Logic	new/old/match/flagged
✔ Redis Update Logic	Safe write to session keys
✔ Kafka Completion Emission	Task completed notification to Entry

10. Interview Preparation Checklist

Skill Area	Demonstrated Capability
Kafka Consumer Logic	Multi-topic flow coordination
Redis State Management	Session TTL handling + atomic updates
AES Cryptography	Decryption pipeline security
Face Vectorization	Real-time ONNX inference & tensor prep
FAISS Matching	Efficient similarity computation
Failure Handling	Logs and structured error output

11. Step-by-Step Development Guide

1. Write Kafka consumer for FOLR_REGISTER_OR_LOGIN
2. Fetch encrypted blob from PostgreSQL using session_id
3. AES decrypt blob and validate image
4. Resize and normalize image using OpenCV
5. Load ONNX model and generate 512-d vector
6. Call FAISS and determine match category

7. Populate Redis: vector, result, score
8. Emit TASK_COMPLETED_FOLR Kafka event
9. Handle corner cases:
    - Invalid image
    - Low confidence
    - Already existing vector (duplicate)
10. Write unit tests for all functional modules

12. Postman & GitHub Guidelines

 REST Not used → No Postman.

GitHub Branch Suggestions:

- feature/onnx-face-embedding
- feature/faiss-face-matcher
- feature/kafka-face-consumer
- feature/redis-result-writer

Testing Tools:

- Kafka CLI or test_producer.py
- Redis CLI for state validation
- ONNX local test script with known faces
- Simulated PostgreSQL blob inserter