Hackathon Task: Intelligent Face Tracker with Auto-Registration and Visitor Counting

---

Objective

Build an AI-driven unique visitor counter that processes a video stream to detect, track, and recognize faces in real-time. The system should automatically register new faces upon first detection, recognize them in subsequent frames, and track them continuously until they exit the frame. Every entry and exit must be logged with a timestamped image and stored both locally and in a database. The ultimate goal is to accurately count the number of **unique visitors** in the video stream.

Core Functional Modules to Build

---

1. Face Detection, Recognition & Tracking Pipeline (Python + YOLO + InsightFace + Database)

- Process video from a provided sample file (**Drive link containing the video: https://drive.google.com/drive/folders/15YCN3CYb97GyIoNUV6NJxGNIf-rUBFUJ?usp=sharing**)
  - ι **During the interview**, this will be tested using a **live RTSP camera stream** instead of the video file.
- Use a **YOLO-based model** (e.g., YOLOv8) for real-time face detection.
- Generate facial embeddings using a **state-of-the-art face recognition model**, such as:
  - **InsightFace**
  - **ArcFace**
  - **Dlib-ArcFace**
  - + **Avoid using the `face_recognition` Python library** — it is not accurate enough for production-grade applications.
- Every **new face** must be:
  - Automatically registered and assigned a unique identifier.
  - Stored in the database with metadata (ID, timestamp).
- Once registered, a face should be **recognized and tracked** across frames using detection + tracking logic.
- Use a configurable parameter from `config.json` to define:
  - How many frames to skip between **detection cycles**.

2. Logging System (Python + Filesystem + Database)

- Every **face entering** and **exiting** the camera frame must generate **exactly one entry** each.
- Log must include:
  - Cropped face image.
  - Timestamp of the event (entry or exit).
  - Event type (entry or exit).
  - Face ID.
- Store cropped face images in a **structured local folder system** (e.g., `logs/entries/YYYY-MM-DD/`).
- Store all metadata in the database.
- A **mandatory log file** (e.g., `events.log`) should track all critical system events including:
  - Face entry, recognition, tracking, and exit.
  - Embedding generation and registration.

---

3. Unique Visitor Counting

- Maintain an accurate count of **unique faces** detected during the stream.
- Re-identification of the same face in later frames **must not** increment the count.
- Count should be retrievable from the database or derived from logs.

Mandatory TechStack

Module	Tech
Face Detection	YOLO (v5/v8 or similar)
Face Recognition	InsightFace / ArcFace / SOTA model
Tracking	OpenCV / DeepSort / ByteTrack
Backend & Processing	Python

Database	Any (SQLite, MongoDB, PostgreSQL)
Configuration	JSON (config.json for detection skip)
Logging	Log file + Local image store + DB
Camera Input	Provided video file (for dev), RTSP stream (for interview)

Evaluation Instructions

This hackathon task is designed to assess your ability to build real-time intelligent systems that combine AI, video processing, and structured logging.

You must:

- Write clean, modular Python code.
- Include a `README.md` file with:
  - Setup instructions.
  - Assumptions made.
  - Sample `config.json` structure.
  - Architecture diagram of your application.
  - A Loom or YouTube video link explaining and demonstrating your solution.**(If you don't submit an explanatory video, your submission will not be reviewed)**
  - Add a line "This project is a part of a hackathon run by https://katomaran.com " in your `README.md` file's bottom

- Submit your github repository in the google form before **12PM Sunday, September 28th, 2025**
- Include sample output from the video file: logs, images, and DB entries.
- Usage of AI tools is encouraged but you must completely understand your generated code. The prompts you used will be evaluated during the interview, so please be prepared accordingly.

Additional Instructions

- Log files are **mandatory** and will be used to verify the correctness of your implementation.
- A frontend is **not required**, but including one will be considered an **added advantage**.

- Code must be modular, scalable, and clearly commented.
- Database and file logging must be consistent and resilient to unexpected interruptions.
- Although we would recommend everyone to follow the instructions above, you may decide to ignore a few for an innovative reason, but you must ensure your submission is good so you get a chance in the next round to explain it to us.

Good luck!