IOT Smart Home Project

1. Introduction

In this project, we designed and implemented an IoT sensing system leveraging a Raspberry Pi equipped with a camera and two sensors (flame and flammable gas sensors) to monitor a user's home. The Raspberry Pi hosts a local server exposing endpoints that return sensor data and the latest camera frame upon request. An external server hosts the Telegram bot and the main application logic. This setup allows users to remotely check the camera feed, monitor sensor values, and receive alerts if an unknown face is detected or if a sensor indicates a dangerous state.
Key features include:

- Real-time face recognition using a trained model.
- Fame and flammable gas sensor integration.
- Local web server on the Raspberry Pi to share sensor data and camera frames.
- External server (Telegram bot + backend) to handle user commands, database operations, and job scheduling.

2. System Architecture

2.1. Hardware Setup

2.1.1. Raspberry Pi

- Connected to a camera for image capture.
- Equipped with flame and flammable gas sensors.
- Exposes a local server to handle frame requests and sensor data.

2.1.2. Power Supply

- Powers the Raspberry Pi, camera, and sensors.

2.1.3. Network Connection

- Raspberry Pi is connected via Ethernet or Wi-Fi to the local network, enabling communication with the external server.

2.2. Software Components

2.2.1. Local Server (on Raspberry Pi)

Exposes endpoints (e.g., /current_frame) to:

- Provide the current camera frame upon request.
- Return the states of flame, and flammable gas sensors (e.g., 0 or 1).

2.2.2. External Server

Hosts the Telegram bot and the core application logic, including:

- Face detection scheduling.
- Sensor information scheduling.
- Notification and alert dispatch to users.
- Communicates with the Raspberry Pi local server for on-demand data (frames and sensor values).
- Interacts with the database to track user connections, face detection, and sensor monitoring states.

2.2.3. Database

Contains two tables: **Chat** and **Bot**.

- **Bot**:
    - BotID (primary key)
    - FaceDetection (boolean)
    - SensorInfo (boolean)
- **Chat**:
    - ChatID (primary key)
    - BotID (foreign key)
    - FaceDetection (boolean)
    - SensorInfo (boolean)

Although we use only one official Telegram bot token, our database design allows for multiple *logical* “Bot” entries—one per household’s Raspberry Pi. Each Chat entry links a user’s chat session to a particular BotID, enabling independent face detection and sensor monitoring settings. This setup makes it easy to scale the system to any number of households without needing to register multiple Telegram bots.

3. Endpoint Details and Workflows

Since there are two servers—one on the Raspberry Pi and one on the external server—this section is split accordingly.

3.1 Local Server (Raspberry Pi)

3.1.1. GET /current_frame

- **Description**: Returns the latest camera frame.
- **Response**: Binary data of the current camera frame.
- **Use Case**: The external server calls this endpoint every time it needs the latest image for face detection.

3.2.2. GET /get_sensor_info

- **Description**: Returns the current values of flame and flammable gas sensors. (also returns CO sensor’s value, however this is not used as it doesn’t work correctly).
- **Response (JSON example)**: {“CO”: 1, "flame": 0, "flammable": 1}
- **Use Case**: The external server polls this endpoint periodically to check sensor status.

3.2.3. GET /set_alarm

- **Description**: Activates an alarm (a buzzer) by toggling the GPIO output pin on and off a certain number of times. This provides a simple auditory warning mechanism when a threat is detected by the system.
- **Response**: None
- **Use Case**: The external server calls this endpoint after detecting a critical event (hazardous sensor readings) to trigger a physical alarm on the Raspberry Pi.

3.2. External Server Endpoints

The external server hosts the Telegram bot and also provides endpoints (if needed) to handle user interactions, scheduling, and database operations. Here is the structure:

3.2.1. /start (Telegram Command)

- **Description**: Initializes the conversation with the Telegram bot.
- **Bot Behavior**: Greets the user and provides basic instructions.

3.2.2. /connect_to_bot [user_key] (Telegram Command)

- **Description**: Links a user's Telegram ChatID to a specific BotID in the database using the provided user_key.
- **Database Action**: Creates or updates a Chat record to reference the correct Bot.

3.2.3. /start_detecting_faces (Telegram Command/Endpoint)

- **Description**: Enables face detection globally for the specified Raspberry Pi (identified by BotID) and starts a recurring job for the user that gives the command.
- **Database Action**:
  - Set Bot.FaceDetection = true for the relevant BotID.
  - Update the user's Chat.FaceDetection = true.
- **Job Scheduling**: A job runs every 2 seconds to call GET /current_frame on the Raspberry Pi, apply face detection, and send alerts if an unknown face is detected.

3.2.4. /start_face_detection_notifs (Telegram Command)

- **Description**: If /startDetectingFaces has already been called so that Bot.FaceDetection is true, this endpoint allows another user connected to the same Raspberry Pi to opt into receiving face detection notifications.
- **Database Action**:
  - Set Chat.FaceDetection = true for the requesting user's chat.
- **Job Scheduling**: Creates a user-specific job.

3.2.5. /stop_face_detection_notifs (Telegram Command)

- **Description**: Stops face detection notifications for the requesting user only.
- **Database Action**:

    - Set Chat.FaceDetection = false for the user’s chat.
- **Job Scheduling**: Removes the job that sends notifications to the user.

3.2.6. /stop_detecting_faces (Telegram Command)

- **Description**: Disables face detection globally for the Raspberry Pi.
- **Database Action**:
    - Set Bot.FaceDetection = false for the relevant BotID.
    - Set all associated Chat.FaceDetection = false.
- **Job Scheduling**: Cancels *all* face detection jobs for every user linked to that Raspberry Pi.

3.2.7. /start_getting_sensor_info (Telegram Command)

- **Description**: Enables getting sensor info globally for the specified Raspberry Pi (identified by BotID) and starts a recurring job for the user that gives the command.
- **Database Action**:
    - Set Bot.SensorInfo = true for the relevant BotID.
    - Update the user’s Chat.SensorInfo = true.
- **Job Scheduling**: A job runs every 2 seconds to call GET /get_sensor_info on the Raspberry Pi, and send alerts if one of the sensors detect something.

3.2.8. /start_sensor_info_notifs (Telegram Command)

- **Description**: If /startGettingSensorInfo has already been called so that Bot.SensorInfo is true, this endpoint allows another user connected to the same Raspberry Pi to opt into receiving sensor info notifications.
- **Database Action**:
    - Set Chat.SensorInfo = true for the requesting user’s chat.
- **Job Scheduling**: Creates a user-specific job.

3.2.9. /stop_sensor_info_notifs (Telegram Command)

- **Description**: Stops sensor info notifications for the requesting user only.
- **Database Action**:
    - Set Chat.SensorInfo = false for the user’s chat.
- **Job Scheduling**: Removes the job that sends notifications to the user.

3.2.10. /stop_getting_sensor_info (Telegram Command)

- **Description**: Disables getting sensor info globally for the Raspberry Pi.
- **Database Action**:
  - Bot.SensorInfo = false for relevant BotID
  - Set all associated Chat.SensorInfo = false.
- **Job Scheduling**: Removes all the jobs that get sensor info from the given Raspberry Pi.

3.2.11. /get_current_frame (Telegram Command)

- **Description**: Gets the latest frame from Raspberry Pi.

3.2.12. /help (Telegram Command)

- **Description**: Shows a help message to the user.

4. Implementation Details

4.1. Local Server Implementation

A lightweight service runs on the home device to capture camera images and read sensor data. It also supports toggling an on-board alarm for critical events. This design offloads computationally heavier tasks (like face recognition) to the external server, keeping the local side minimal.

4.1.1 Key Functional Components

- **Camera Capture:** Provides single snapshots from the device's camera when requested.
- **Sensor Reading:** Monitors various pins for flame, gas, or other signals, returning 0 or 1 to indicate status.
- **Alarm Toggle:** Activates a buzzer in short intervals, triggered upon receiving a signal from the external server.

4.1.2 Overall Flow

- **Request Handling:** The service receives on-demand requests (e.g., "provide the latest image" or "get current sensor readings") from the external server.

- **Hardware Interaction:** It gathers data from the camera and sensors, or toggles an alarm pin as instructed.
- **Response:** Processed information (image bytes or sensor statuses) is sent back, or an alarm is briefly switched on and off, completing the request.

4.2. External Server Implementation

4.2.1. Telegram Bot Library

The external server runs a Telegram bot using the python-telegram-bot library, orchestrating user commands and scheduling tasks for face detection and sensor monitoring:

4.2.1.1. Key Functional Components

- **Command Handlers**: Commands like /start and /start_detecting_faces update the database and schedule or stop recurring jobs.
- **JobQueue**: Periodically polls the Pi for frames (face detection) or sensor readings (CO, flame, flammable gas), sending alerts or activating the Pi's alarm as needed.
- **Database Integration**: Each user's ChatID links to a BotID; toggling detection or sensor info updates both `Chat` and `Bot` flags to manage multi-user, multi-bot environments.

4.2.2.2. Overall Flow

- **User Command** → Bot updates DB → Creates or removes a scheduled job.
- **Scheduled Job** → Calls Pi endpoints → Processes frames/sensor data → Sends messages if necessary.
- **Database** ensures each user-bot pair is tracked correctly, enabling multiple households to use a single Telegram bot token and multiple people in the same house setting different preferences.

4.2.2. Face Recognition System

This part outlines the development of our face recognition system, designed to detect and recognize faces efficiently to understand whether an unknown person is in the house. The system leverages the capabilities of the Mediapipe library for face detection and the face_recognition API for face encoding and recognition.

Face Detection

The face detection process is implemented using Mediapipe, a versatile library developed by Google. Mediapipe's Face Detection module offers high accuracy and real-time performance, making it suitable for detecting faces in various lighting and pose conditions. The key steps are as follows:

- **Input Image Acquisition:** Images from the database are fed into the Mediapipe pipeline.
- **Face Detection:** Mediapipe detects facial landmarks and bounding boxes.
- **Region of Interest (ROI):** The detected face regions are cropped for further processing

Face Encoding

Once the regions are detected, the face_recognition API encodes the faces. This step involves:

- **Feature Extraction:** The face_recognition API computes a 128-dimensional embedding vector for each detected face, capturing unique facial features.
- **Database Creation:** These embeddings are stored in a pickle file, indexed by corresponding labels (e.g., person names or IDs).

Face Recognition

During the face recognition phase, Mediapipe first detects faces in the input image or video frame. The detected faces are then passed to the face_recognition API for comparison with the stored embeddings. The recognition process includes:

- **Face Encoding:** Encoding the detected faces using the same method as during database creation
- **Comparison:** Computing the Euclidean distance between the input face embeddings and the database embeddings stored in the pickle file.
- **Decision:** Assign the label of the closest match if the distance is below a predefined threshold.

Implementation

The system is implemented in Python using the following libraries:

- **Mediapipe**: For real-time face detection.
- **face_recognition**: For face encoding and recognition.
- **pickle**: For storing and loading face embeddings.

Workflow

1. **Database Preparation**: Images of individuals are collected and processed to generate a database of face embeddings, which are stored in a pickle file.
2. **Detection and Recognition**:
    - The input image or video frame is processed using Mediapipe to detect faces.

    - Detected faces are encoded and matched against the embeddings stored in the pickle file.

5. Conclusion

In this project, we developed an IoT-based home monitoring system that combines a Raspberry Pi local server for data acquisition (camera frames, sensor states) with an external server handling face recognition and user interactions through a Telegram bot. By splitting hardware-focused tasks (sensor readings, alarm toggling) from heavier computations (face detection, scheduling), the system remains modular and scalable. Multiple households can share a single bot token, while the database structure ensures each user’s preferences are managed independently. This design provides real-time notifications and responsive alerts for potential intruders or hazardous conditions, all with minimal overhead on the Raspberry Pi. Future enhancements could include more sensors, improved recognition models, and additional security features to further strengthen system reliability.