TELEPHONY PROJECT

PRESENTATION

PRESENTED BY: Ali Abdallah – 6352
Hassan Mohsen – 6295
Alaa Fadlallah – 6445
Mohammad Kansoun - 6123

AGENDA

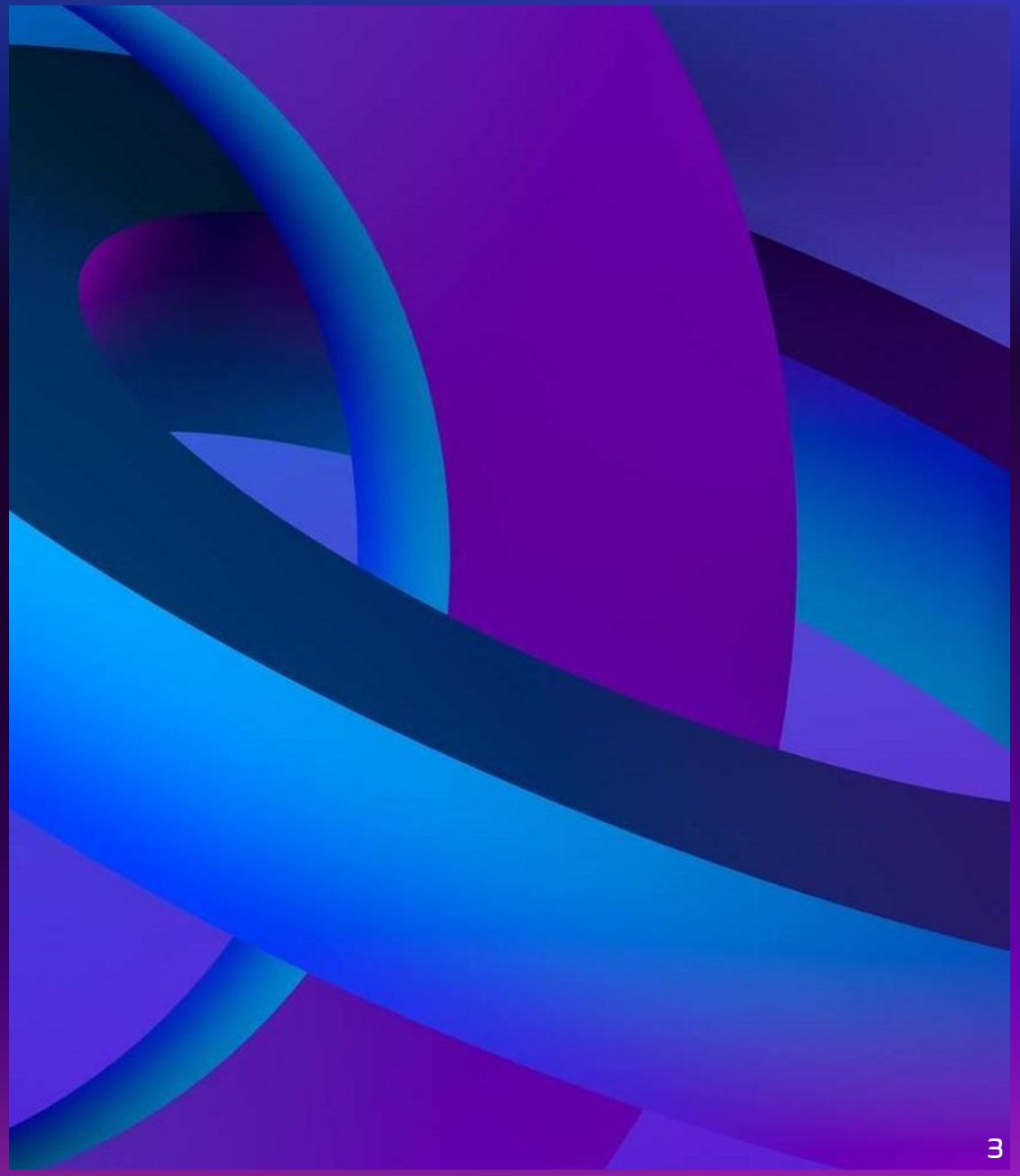
1. Introduction
2. Problem Statement
3. System and Architecture
4. The tool behind the project
5. limitations
6. Future work

1. INTRODUCTION:

1.WITH GROWING SECURITY CONCERNS, THE DEMAND FOR INTELLIGENT, RESPONSIVE SYSTEMS IS MORE CRITICAL THAN EVER.

2. OUR SOLUTION COMBINES REAL-TIME FACE RECOGNITION WITH INSTANT MESSAGING TO SECURE ENTRY POINTS.

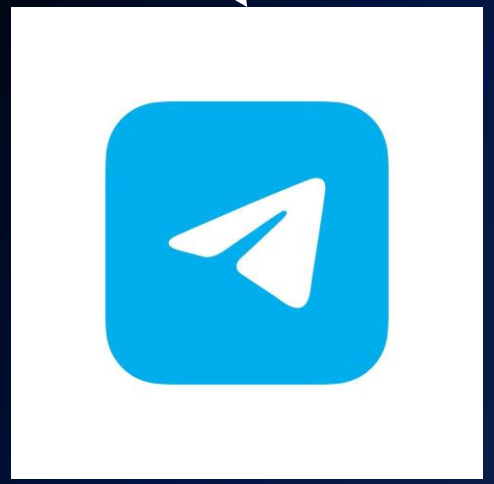
3. BY LEVERAGING A RASPBERRY PI FOR IMAGE CAPTURE AND A LAPTOP FOR PROCESSING, IT ENABLES IMMEDIATE ALERTS AND ENSURES ONLY AUTHORIZED ACCESS.



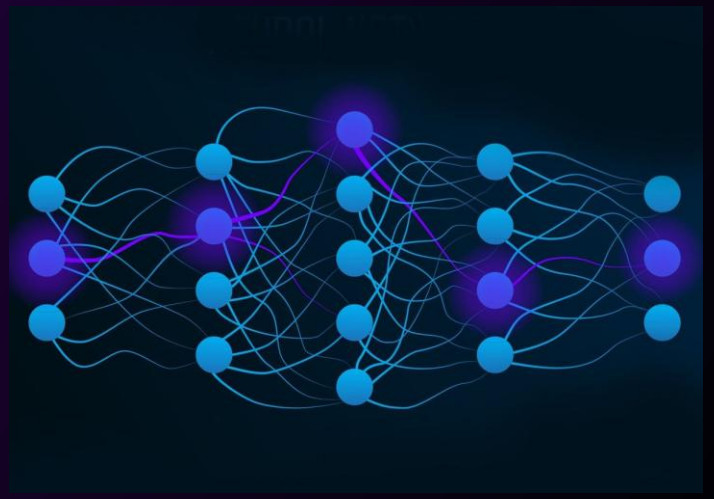
2. PROBLEM STATEMENT

- Legacy systems are unable to identify individuals in real time
- Relying on manual surveillance is time-consuming and unreliable
- Unauthorized entries often go undetected without immediate alerts
- There's a growing demand for intelligent, affordable security solutions

3. SYSTEM ARCHITECTURE



4. FACE RECOGNITION



$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_{128} \end{bmatrix} \in \mathbb{R}^{128\times 1}$$

CODE SNIPPETS

CLIENT CODE (PI)

```python
SERVER_IP = '192.168.0.107'  # Replace with your laptop's IP
SERVER_PORT = 5001

def capture_and_send():
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("❌ Camera not accessible.")
        return

    last_sent = time.time()

    while True:
        ret, frame = cap.read()
        if not ret:
            print("❌ Failed to capture image.")
            continue

        cv2.imshow("Live Camera Feed", frame)

        if time.time() - last_sent >= 2.0:
            _, img_encoded = cv2.imencode('.jpg', frame)
            img_bytes = img_encoded.tobytes()
            img_size = len(img_bytes)

            try:
                with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                    s.connect((SERVER_IP, SERVER_PORT))
                    s.sendall(img_size.to_bytes(4, byteorder='big'))
                    s.sendall(img_bytes)
                    print("📤 Frame sent.")
            except Exception as e:
                print(f"🔌 Connection error: {e}")

            last_sent = time.time()

        # Quit on 'q' key
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

```

Steps:

1.Initialize the Camera

Set up a cv2.VideoCapture object to access the camera stream.

2.Capture Video Frames Continuously

Read frames from the camera in a loop to keep the video stream active.

3.Send a Frame to the Server Every 2 Seconds

Use a timer or delay mechanism to transmit one frame every 2 seconds to the server.

CODE SNIPPETS

SERVER CODE

```python
def start_server():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((HOST, PORT))
        server.listen()

        print(f"📡 Server listening on {HOST}:{PORT}...")

        while True:
            conn, addr = server.accept()
            with conn:
                print(f"📩 Connection from {addr}")

                size_data = conn.recv(4)
                if not size_data:
                    continue

                img_size = int.from_bytes(size_data, byteorder='big')
                img_bytes = b''
                while len(img_bytes) < img_size:
                    chunk = conn.recv(min(4096, img_size - len(img_bytes)))
                    if not chunk:
                        break
                    img_bytes += chunk

                np_arr = np.frombuffer(img_bytes, np.uint8)
                frame = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)

                if frame is not None:
                    handle_received_frame(frame)
                else:
                    print("❌ Failed to decode received image.")
```

Steps:

1. Receive image from raspberry pi
2. Extract the size of the image
3. Transfer the image into a cv2 frame
4. Pass the frame to:
   `handle_received_frame`

CODE SNIPPETS

HANDLING FRAME

```python
def handle_received_frame(frame: np.ndarray):
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    locations = face_recognition.face_locations(rgb)
    encodings = face_recognition.face_encodings(rgb, locations)

    for (top, right, bottom, left), face_encoding in zip(locations, encodings):
        matches = face_recognition.compare_faces(known_encodings, face_encoding, tolerance=0.5)
        name = "Unknown"

        if True in matches:
            best_match = matches.index(True)
            name = known_names[best_match]

            if name not in seen_faces:
                seen_faces.add(name)

                try:
                    print("sending name ...")
                    message = generate_visitor_message(name)
                    bot.send_message([message])
                except Exception as e:
                    print(f"⚠️ Failed to send message: {e}")

                try:
                    face_image = frame[top:bottom, left:right]
                    face_rgb = cv2.cvtColor(face_image, cv2.COLOR_BGR2RGB)
                    pil_image = Image.fromarray(face_rgb)
                    print("sending image ...")
                    bot.send_pil_image(pil_image)
                except Exception as e:
                    print(f"⚠️ Failed to send image: {e}")

        # Optional: draw on frame (for debugging)
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
        cv2.putText(frame, name, (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2)
```

Steps:

1.Identify the Person from Face Embeddings

Compare the incoming face encoding against the known encodings to retrieve the corresponding name.

2.Prevent Duplicate Alerts

Add the identified name to a `**seen_faces** set to avoid repeated notifications. Optionally, implement a timer-based cooldown before allowing re-alerts.

3.Send Notification via Telegram Bot

Use the Telegram bot to send the person's name along with the captured face image to the designated chat.

5. LIMITATIONS

1. Limited Camera Performance

USB webcams may have low resolution or frame rate, affecting face detection accuracy, especially in poor lighting conditions.

2. Face Recognition Accuracy

The `face_recognition` library may struggle multiple things.

3. Basic spoofing if no liveness detection

3. Lack of Real-Time Efficiency

Sending one frame every 2 seconds may miss fast-moving subjects or brief appearances.

6. FUTURE WORK

1. Develop an Administrative Dashboard for Face Database Management

Create a secure and user-friendly web-based dashboard that allows administrators to add, update, and remove entries from the face database.

2. Enable Multiple Face Tracking

Track and label multiple faces simultaneously, sending alerts for each one independently.

3. Improve Face Recognition Model

Replace `face_recognition` with more advanced deep learning models (e.g. FaceNet, ArcFace) for higher accuracy and robustness.