```
import cv2
import time
from keras.models import load_model
import numpy as np
from datetime import datetime, timedelta



#Facial Recognition with FaceNet
#Blurriness and Lighting Check


# Load pre-trained FaceNet model
facenet_model = load_model('facenet_keras.h5')

# Function to compute embeddings
def get_embedding(model, face_pixels):
    face_pixels = face_pixels.astype('float32')
    mean, std = face_pixels.mean(), face_pixels.std()
    face_pixels = (face_pixels - mean) / std
    sample = np.expand_dims(face_pixels, axis=0)
    return model.predict(sample)[0]

# Blurriness Detection
def is_blurry(image, threshold=100):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    laplacian_var = cv2.Laplacian(gray, cv2.CV_64F).var()
    return laplacian_var < threshold

# Low Light Detection
def is_low_light(image, threshold=50):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return gray.mean() < threshold

# Attendance Logging
attendance_log = {}

def log_attendance(name, confidence):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    attendance_log[name] = {"timestamp": timestamp, "confidence": confidence}
    print(f"Attendance logged for {name} with {confidence*100:.2f}% confidence at {timestamp}.")



#Real-Time Face Presence Tracking
#Multiple Face Detection
#Suspicious Behavior Detection



def start_test_proctoring(face_database, session_duration=16*60*60):
    cap = cv2.VideoCapture(0)
    start_time = datetime.now()
    end_time = start_time + timedelta(seconds=session_duration)

    no_face_start_time = None
    face_absence_limit = 30  # seconds
    suspicious_events = []

    while datetime.now() < end_time:
        ret, frame = cap.read()
        if not ret:
            print("Failed to capture frame. Exiting...")
            break

        faces = detect_face(frame)  # Detect faces
        face_present = len(faces) > 0

        if face_present:
            # Reset absence timer
            no_face_start_time = None

            # Check multiple faces
            if len(faces) > 1:
                suspicious_events.append("Multiple faces detected")
                cv2.putText(frame, "ALERT: Multiple faces detected!", (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

            # Recognize each face and check for quality
            for face in faces:
                x, y, w, h = face
                face_img = frame[y:y+h, x:x+w]
```

```
                face_img_resized = cv2.resize(face_img, (160, 160))

                # Check for image quality
                if is_blurry(face_img):
                    cv2.putText(frame, "Blurry Image", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 255), 2)
                    continue
                if is_low_light(face_img):
                    cv2.putText(frame, "Low Light", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 255), 2)
                    continue

                # Recognize face
                embedding = get_embedding(facenet_model, face_img_resized)
                recognized = False
                for name, db_embedding in face_database.items():
                    similarity = np.dot(embedding, db_embedding) / (
                            np.linalg.norm(embedding) * np.linalg.norm(db_embedding))
                    if similarity > 0.5:
                        recognized = True
                        cv2.putText(frame, f"Recognized: {name}", (x, y - 10),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
                        break

                if not recognized:
                    cv2.putText(frame, "Unknown", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2)
        else:
            # Track absence of face
            if no_face_start_time is None:
                no_face_start_time = time.time()
            elif time.time() - no_face_start_time > face_absence_limit:
                suspicious_events.append("Face absent for more than 30 seconds")
                cv2.putText(frame, "ALERT: No face detected!", (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

        # Display the frame
        cv2.imshow("Test Proctoring", frame)

        # Exit condition
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    print("Test session ended.")
    print("Suspicious Events Log:")
    for event in suspicious_events:
        print(event)



#Utility Function for Face Detection
def detect_face(image):
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    return faces



Start coding or generate with AI.
```