Face Authentication & Attendance System - Detailed Technical Report

1. Project Overview

This project is a secure, biometric attendance system that uses Facial Recognition to authenticate users. It is designed to be robust against simple spoofing attacks (like holding up a photo) by implementing **Liveness Detection** via eye-blink analysis.

The system is built as a web application, allowing users to register their face and "Punch In/Out" using any device with a camera.

Key Features

- **Facial Recognition**: Uses the LBPH (Local Binary Patterns Histograms) algorithm for recognizing registered users.
- **Liveness Detection**: Uses **MediaPipe Face Mesh** to detect eye blinks, ensuring the user is a real person and not a static image.
- **Web Interface**: A modern, responsive UI for easy interaction.

- **JSON Database**: Lightweight, file-based storage for user data and attendance logs.

---

2. System Architecture

The system follows a standard Client-Server architecture. The Client (Browser) captures video frames, and the Server (Python/FastAPI) processes them using Computer Vision algorithms.

```
Parse error on line 6:
...        subgraph "Backend Processing"
---------------------^
Expecting 'SEMI', 'NEWLINE', 'SPACE', 'EOF', 'GRAPH', 'DIR',
'TAGEND', 'TAGSTART', 'UP', 'DOWN', 'subgraph', 'end', 'SQE',
'PE', '-)', 'DIAMOND_STOP', 'MINUS', '--', 'ARROW_POINT',
'ARROW_CIRCLE', 'ARROW_CROSS', 'ARROW_OPEN',
'DOTTED_ARROW_POINT', 'DOTTED_ARROW_CIRCLE',
'DOTTED_ARROW_CROSS', 'DOTTED_ARROW_OPEN', '==',
'THICK_ARROW_POINT', 'THICK_ARROW_CIRCLE', 'THICK_ARROW_CROSS',
'THICK_ARROW_OPEN', 'PIPE', 'STYLE', 'LINKSTYLE', 'CLASSDEF',
'CLASS', 'CLICK', 'DEFAULT', 'NUM', 'PCT', 'COMMA', 'ALPHA',
'COLON', 'BRKT', 'DOT', 'PUNCTUATION', 'UNICODE_TEXT', 'PLUS',
'EQUALS', 'MULT', got 'STR'
```

---

3. Technical Implementation Details

3.1. Tech Stack

- **Language**: Python 3.10+
- **Web Framework**: FastAPI (High performance, async support)
- **Computer Vision**:
    - **OpenCV**: For image processing and the LBPH Face Recognizer.
    - **MediaPipe**: Google's ML solution for high-fidelity Face Landmarking (used for liveness).
- **Frontend**: HTML5, JavaScript (Vanilla), CSS3.

3.2. Liveness Detection (Anti-Spoofing)

Goal: Prevent users from logging in using a photo of another person. **Method**: **Blink Detection**.

1. The frontend captures a **burst of 10 images** over ~1.5 seconds.
2. The backend calculates the **Eye Aspect Ratio (EAR)** for each frame.
3. EAR is a geometric measurement of how "open" the eye is.
   - High EAR (> 0.22) = Open Eyes.
   - Low EAR (< 0.18) = Closed Eyes (Blink).
4. **Verification**: The system confirms liveness ONLY if it detects both an "Open" state and a "Closed" state within the image sequence.

```
# Pseudo-code for Decision Logic
if (min_EAR < CLOSED_THRESHOLD) AND (max_EAR > OPEN_THRESHOLD):
    return "LIVE (Blink Detected)"
else:
    return "SPOOF (Static Face)"
```

3.3. Face Recognition

Method: **LBPH (Local Binary Patterns Histograms)**.

- LBPH is robust to local lighting changes.
- It divides the face into small cells, compares each pixel to its neighbors, and builds a histogram.
- During recognition, it compares the histogram of the input face to the training data.
- **Lighting Enhancement**: Before processing, we apply **CLAHE** (Contrast Limited Adaptive Histogram Equalization) to improve accuracy in poor lighting.

---

4. Workflows & Usage

4.1. Registration Process

- **Step 1**: User enters their name.
- **Step 2**: Sits in front of the camera.
- **Step 3**: System captures 10 training images.

- **Step 4**: The model is immediately retrained to include the new user.

> **[INSERT SCREENSHOT HERE: Registration Page showing camera feed and name input]**

4.2. Attendance "Punch In"

- **Step 1**: User clicks "Punch IN".
- **Step 2**: System prompts: *"Look at the camera and BLINK"*.
- **Step 3**: User blinks naturally. System captures the frame sequence.
- **Step 4**: Backend verifies the blink, identifies the user, and logs the time.

> **[INSERT SCREENSHOT HERE: Main Dashboard with 'Punch In' success message]**

4.3. Viewing History

- **Step 1**: User navigates to the History page.
- **Step 2**: A table displays all clock-in/out events with timestamps.

> **[INSERT SCREENSHOT HERE: History Log Table]**

---

5. Project Structure Overview

- **`main.py`**: The entry point. Handles all web requests (API endpoints) and connects the frontend to the core logic.
- **`core/auth.py`**: The "Brain". Contains the `FaceAuthSystem` class which handles:
  - `check_liveness()`: The blink detection logic.
  - `identify_user()`: The face recognition logic.
  - `train_model()`: Updates the AI model when a new user joins.
- **`core/db.py`**: Handles reading/writing to `data.json`.
- **`static/`** & **`templates/`**: Contains the customized, responsive User Interface.
- **`data/`**: Stores the raw face images, the trained model (`trainer.yml`), and the database.

---

6. Future Improvements

- **Head Pose Detection**: Add "Turn Head Left/Right" challenges for even higher security.
- **SQL Database**: Migrate from JSON to SQLite or PostgreSQL for scalability.
- **User Management**: Add Admin panel to delete or manage users.