Smart Classroom Attendance Automation Using Facial Recognition

Malla Jaswanth
Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
j.malla@iitg.ac.in

***Abstract*—This paper presents a digital attendance system using face recognition to enhance transparency and efficiency in classroom environments. The system captures a facial image, verifies if the person is registered, and marks attendance automatically. Developed with FastAPI as the backend and React for the frontend. The application enables real-time attendance tracking, reduces manual effort, prevents proxy attendance, and provides a scalable solution.**

I. Introduction

Traditional methods of recording attendance, such as roll calls or paper-based sign-in sheets, are often time-consuming and inefficient, particularly in classrooms with a large number of students. These manual approaches are prone to human error and are susceptible to proxy attendance, where one student marks the presence of another. Such practices compromise the integrity of the attendance data and reduce the reliability of the system.

To address these challenges, we propose a smart attendance mechanism that relies on real-time face recognition, offering contactless, non-intrusive identity verification. In the digital era, face recognition technology has become an integral part of various sectors due to its versatility and convenience. The system captures facial images of individuals, verifies them against a pre-registered database, and marks their attendance automatically. This approach ensures accurate, tamper-proof, and efficient tracking of student presence, reducing manual effort and improving transparency.

The proposed system is built using FastAPI for the backend, Python and React for the frontend, ensuring real-time performance, easy scalability, and a smooth user experience. By leveraging modern web technologies and facial recognition models, this system performs accurate student recognition and logs the attendance seamlessly, ensuring data integrity and preventing repeated or fraudulent entries.

II. System Overview

The proposed system functions in two main modes: **Registration** and **Attendance**. During registration, a student's facial features are captured and stored. During attendance, the system compares the captured image against the stored encodings to validate identity. A student is allowed to mark their presence only once per day, ensuring fairness and preventing misuse.

The process begins with student registration, where each student's face is captured and stored in the system along with their name and a unique identifier. This step ensures that every registered face is unique and linked to a single identity.

Once registered, students can mark their attendance using a webcam interface. When a student looks at the webcam, the system captures a live image and compares it with the stored facial data using facial recognition algorithms. If a match is found and attendance has not already been marked for the day, the system records the attendance in a CSV file along with the date and time.

The system also includes safeguards to prevent multiple faces from being processed at once and to avoid duplicate registrations with different names. If a face is already in the system, the student cannot register again under a different identity.

III. Technologies Used

A. *Frontend*

The frontend of the application is built using **React.js**, a powerful JavaScript library for building dynamic and responsive user interfaces. React efficiently manages UI components, user interactions, and state transitions, making the application fluid and user-friendly.

The frontend consists of **four main components**:

- **Home.js** – Serves as the landing page of the application. It provides a starting point for users and allows them to begin interacting with the system.
- **ModeSelect.js** – Allows users to choose between two available modes: *Registration* or *Attendance*. This ensures that users clearly select the intended action before proceeding.
- **Registration.js** – Handles the user registration process. Users are required to enter their name, and only after doing so, the application prompts for webcam access to capture a face image. If a user attempts to proceed without entering a name, an error message is displayed, guiding them to complete the required field.
- **Attendance.js** – Facilitates the attendance marking process. It activates the webcam to capture a live image of the user, which is then sent to the backend for face recognition and verification.

To enable webcam functionality, the application utilizes **react-webcam**, a React package that provides seamless access to the user's webcam for real-time image capture.

Styling across the application is handled using **CSS**, ensuring a clean, consistent, and responsive design throughout the user experience.

B. Backend

The backend of the application is developed using FastAPI, a modern Python web framework known for its speed and support for asynchronous operations. It handles API requests for registering users and marking attendance through face recognition.

Technologies & Tools Used

- **FastAPI** – Web framework for building RESTful APIs (register, attendance).
- **face_recognition** – For face detection, encoding, and comparison.
- **NumPy** – For handling and comparing face encoding vectors.
- **OpenCV (cv2)** – For image processing, debugging, and drawing bounding boxes.
- **CSV module** – To store attendance data in a CSV file.
- **CORS Middleware** – Enables communication between frontend and backend by resolving cross-origin issues.

File Structure

- **main.py** – Contains the core FastAPI application with endpoints for user registration and attendance marking.
- **attendance_logger.py** – Utility module for logging attendance and checking if a user has already marked attendance on the current day.
- **faces/** – Directory that stores face encodings in `.npy` format for each registered user.
- **attendance_log.csv** – Automatically generated CSV file that logs attendance records with Name, Timestamp, and Status.
- **debug_images/** – Directory that stores debug images with rectangles drawn around detected faces, used for troubleshooting and verification.

IV. API Endpoints

A) POST /register/

Description:

Registers a new user with a unique name and their facial encoding. Accepts two form fields:

- `name` – User's name.
- `file` – Image file (face photo).

Image Processing:

- Used `face_recognition.load_image_file()` to load the image into a NumPy array.
- Used `face_recognition.face_locations()` and `face_recognition.face_encodings()` to detect and encode faces.
- Built a utility function `save_debug_image()` to draw rectangles around detected faces and save the image as `debug_register.jpg` for debugging purposes.

Validation and Registration:

- Used `len(encodings)` to determine the number of detected faces and return an error if more than one face is present.
- Implemented `is_face_already_registered()` to check if the face already exists in the database by comparing encodings using Euclidean distance.

Summary of Flow:

1) Receive `name` and `image` from the client.
2) Load and process the image.
3) Ensure exactly one face is present.
4) Check if the face is already registered.
5) If not already registered, save the face encoding with the given name.
6) Return a success or relevant error message.

B) POST /attendance/

Marks attendance for a recognized user. Accepts one form field:

- `file` – Image file (face photo).

Core Logic:

- Iterates over `.npy` files in the `faces/` directory which store face encodings.
- If no encodings exist, returns an error instructing the user to register first.
- `face_recognition.face_distance()` to compute the Euclidean distance between the captured encoding and all stored encodings.
- `already_marked_today()` - check if this user already marked attendance for the current date.
- If not already marked, logs attendance using `log_attendance()` which writes to `attendance_log.csv`.

Summary of Flow

1) Read image → detect face → encode.
2) Check for 1 face only.
3) Compare with known encodings in `faces/`.
4) If match found under threshold → log attendance (if not already logged).
5) Otherwise → return appropriate error message.

V. Step-By-Step Process

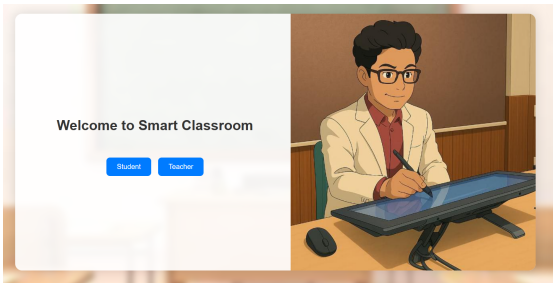


Fig. 1. Home page

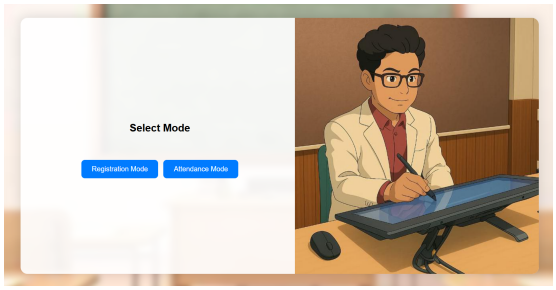


Fig. 2. Mode selection page

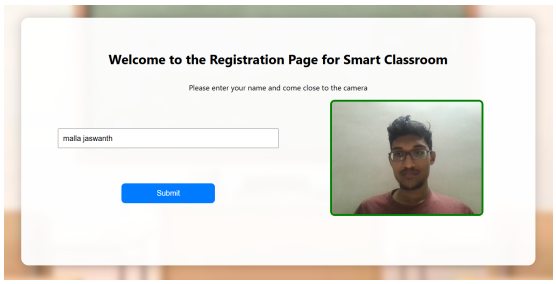


Fig. 3. User registers through his name and facial image

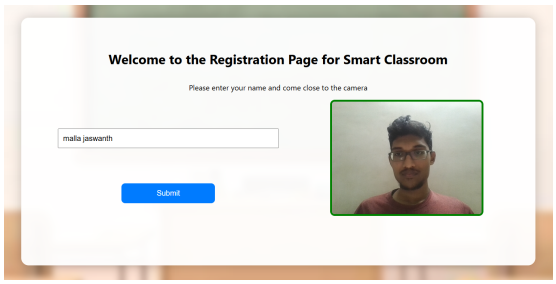


Fig. 4. User registers through his name and facial image

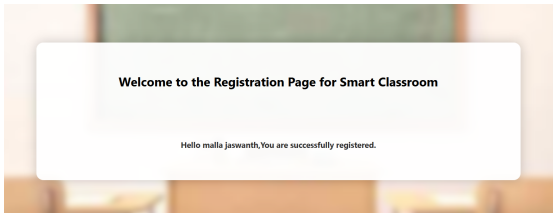


Fig. 5. User sucessfully registered.

After registration when user selects attendance mode for marking attendance,

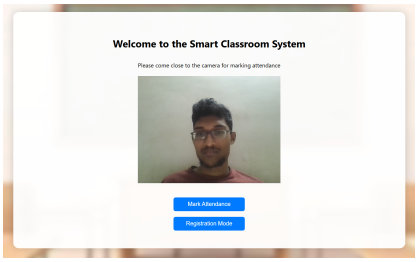


Fig. 6. The system verifies the user's facial data for authentication.

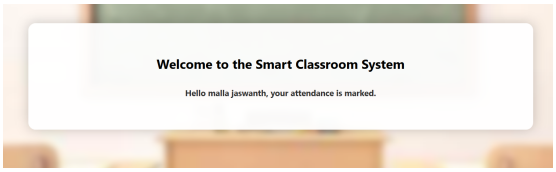


Fig. 7. User successfully marked the attendance if he registers

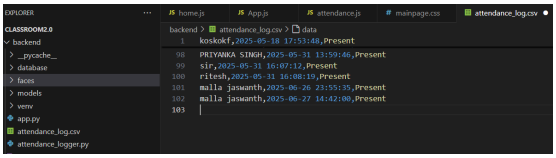


Fig. 8. Attendance.csv files stores attendance

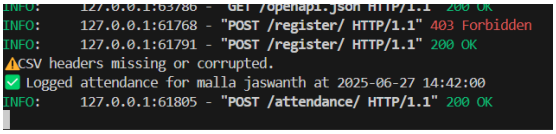


Fig. 9. Terminal output

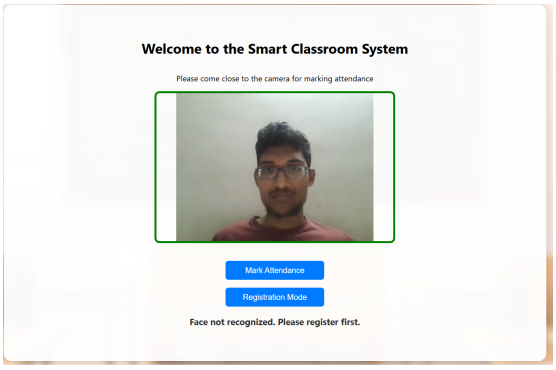
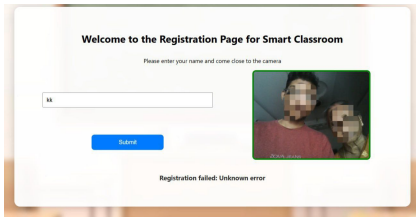


Fig. 10. If user tries to mark attendance without registration

When multiple faces are detected,



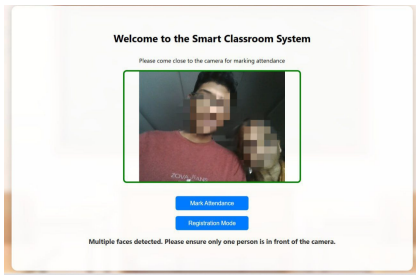


Fig. 11. Multiple faces not allowed

When a user attempts to register with another name, the system will not allow it.

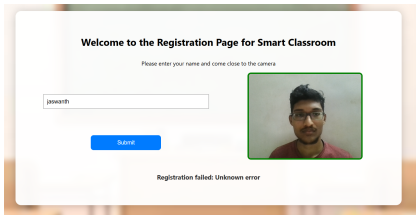


Fig. 12. Unique face per person allowed for registration

VI. Limitations

- Recognition performance can degrade under poor lighting.
- No built-in liveness detection, so spoofing via photographs is possible.
- A fixed face distance threshold might not perform optimally across different environments.

VII. Prospective Enhancements

- Cloud Integration: Storing attendance logs in a centralized cloud database (e.g., Firebase, MongoDB Atlas) for better accessibility and backup.
- Liveness Detection: Adding features to detect spoofing attempts
- Multi-Face Detection: Supporting simultaneous recognition of multiple students in group scenarios (e.g., scanning an entire classroom in one frame).
- Mobile App Support: Developing a mobile application to allow attendance marking via smartphone cameras.

VIII. Conclusion

This paper presented a Smart Classroom Attendance System that leverages facial recognition to automate the process of marking attendance. By integrating a webcam-based image capture mechanism with a FastAPI backend and a React frontend interface, the system ensures quick, contactless, and secure identification of students. The use of the face_recognition library enables accurate face matching, while the modular architecture ensures ease of deployment and maintenance.

The system significantly reduces manual intervention, eliminates proxy attendance, and provides real-time updates, making it a practical solution for modern educational environments. The evaluation shows that the system performs reliably under normal lighting and camera conditions, making it suitable for classroom

Acknowledgment

This work was carried out under the guidance of Prof. Anirban Dasgupta and the supervision of Dhruvika Verma at the Department of Electronics and Electrical Engineering, Indian Institute of Technology (IIT) Guwahati, in pursuit of practical advancements in the field of smart systems

References

[1] React-dev blog - https://react.dev/blog/2023/03/16/introducing-react-dev
[2] w3schools CSS reference - https://www.w3schools.com/CSSref/index.php
[3] FastAPI Documentation - https://fastapi.tiangolo.com/
[4] OpenCV Documentation - https://docs.opencv.org/
[5] NumPy Documentation - https://numpy.org/doc/
[6] Face Recognition Library - https://github.com/ageitgey/face_recognition