

# Smart Campus Attendance: A Three-Factor Biometric IoT System with Firebase-Powered Role-Based Web Dashboards

Vansh Tyagi<sup>1</sup>, Shitanshu<sup>2</sup>, Raj Singh<sup>3</sup>

<sup>1,2,3</sup>B.Tech Scholar, Dept. of Computer Science and Engineering,  
Raj Kumar Goel Institute of Technology, Ghaziabad, India

\*\*\*

**Abstract** - Colleges in India still mark student attendance on paper or with basic card-swipe terminals. Both methods share one fundamental problem: a student can be marked present without being physically there. We built a system to make that impossible. Our Smart Campus Attendance System runs on an ESP32 microcontroller and requires three sequential identity checks before logging any attendance event: a tap of an RFID card, a fingerprint scan, and a face match against the enrolled photograph. All three must pass. A single failure aborts the process and logs a rejected attempt for administrative review. Once verified, the event travels over Wi-Fi to a Firebase Firestore backend, which persists the record and triggers a Socket.IO broadcast. Four separate React 19 dashboards, one each for students, faculty, Heads of Department, and parents, receive this update within milliseconds. Students can check their own attendance percentage and marks, compare their standing against classmates, and read messages from teachers. Faculty can see which students are falling behind, enter marks, and send notifications to parents. The HOD view covers department-wide trends. Parents see only their ward's data, in real time. We tested the system across 50 enrolled students over 312 attempts. Three-factor verification passed at a combined rate of 94.2%. Average end-to-end latency from card tap to dashboard update was 6.7 seconds. Socket.IO propagation to open clients averaged 480 milliseconds. The full stack is open source: Python, React 19, Tailwind CSS, Recharts, Firebase, and Socket.IO.

**Key Words:** IoT, ESP32, RFID, fingerprint recognition, face recognition, Firebase Firestore, Socket.IO, React 19, attendance management, academic ERP, role-based dashboard

## 1. INTRODUCTION

Walk into any undergraduate classroom in India during attendance time and the scene is the same: the teacher calls names, students say present, someone marks a register. It takes ten minutes and produces a record that is easy to manipulate. A student who skips class simply arranges for a friend to answer. By the time a parent finds out their child has 40% attendance, the semester is nearly over.

Card-based systems were introduced to speed this up. They work for speed but not for fraud. Owing a card is not the same as being the person whose name is on it. Single-factor systems share this weakness: defeating one factor defeats the system entirely.

We designed a three-factor terminal to remove that option. Our device, built on an ESP32 microcontroller, requires a student to tap their RFID card, pass a fingerprint match, and clear a face verification check in that order before any attendance event is recorded. The ESP32 costs under 300 rupees, the RC522 RFID module under 50, and the AS608 fingerprint sensor under 150. What has been missing from the literature is a system combining these factors with a real-time data layer and role-specific dashboards that deliver verified data to every stakeholder immediately.

The primary contributions of this work are:

- A three-factor biometric attendance terminal on ESP32 (RFID, fingerprint, face) that removes proxy attendance without expensive infrastructure.
- A Firebase Firestore and Socket.IO pipeline propagating verified events to dashboards in under 500 milliseconds.
- Four role-differentiated React 19 dashboards, Student, Faculty, HOD, and Parent, each scoped to what that user needs.
- A Python face verification service using the face\_recognition library (dlibResNet backend) integrated into the hardware pipeline.

## 2. RELATED WORK

### 2.1 RFID-Based Systems

Umar et al. [1] combined a GSM module with an RFID reader so that when a student swiped a card, the parent received an SMS. Useful for notification, but the system trusted whoever held the card. Patil et al. [2] stored RFID attendance in a structured database, though card sharing remained an unresolved vulnerability.

## 2.2 Fingerprint Recognition

Shoewu and Idowu [3] achieved near-zero false acceptance in a university fingerprint deployment. Their False Rejection Rate rose when sensor surfaces were dirty or students had dry or injured fingertips. Their conclusion was that a backup modality would make any fingerprint-based system more robust in practice.

## 2.3 Face Recognition

Adjabi et al. [4] reviewed a decade of face recognition literature and found that controlled environment 2D systems perform well, but performance drops when ambient conditions change. Guo and Zhang [5] surveyed over 330 deep learning contributions and documented CNN architectures achieving above 97% accuracy on standard benchmarks, while illumination sensitivity remains a deployment constraint.

## 2.4 Multi-Factor and IoT Approaches

Bhatt and Bhatt [6] paired RFID with face recognition for campus access and showed that two factors raise the cost of impersonation considerably. Their system lacked any dashboard layer. Sharma et al. [7] deployed an ESP8266 with RFID and cloud sync, demonstrating IoT-based capture is viable, but without biometric verification or any stakeholder-facing interface.

## 2.5 Gap in Literature

No published system identified in this review combines three biometric factors on IoT hardware with a real-time cloud pipeline and multiple role-differentiated dashboards in a single deployment. We address all three simultaneously.

# 3. SYSTEM ARCHITECTURE

## 3.1 Three Layers

The system has three layers: Hardware Verification, Backend Data, and Presentation. Attendance events originate at the hardware terminal, flow through the backend for persistence and broadcast, and are received in real time by the role-appropriate dashboard. Each layer is decoupled at its interface. The ESP32 communicates with the backend only through HTTPS REST calls. The backend emits Socket.IO events without knowing how many clients are connected. The frontend consumes typed payloads without depending on hardware implementation details.

## 3.2 Hardware Terminal

Each terminal is built around the ESP32 microcontroller, chosen for its dual-core Xtensa LX6 processor, 520 KB of SRAM, and native 802.11 b/g/n Wi-Fi. Four peripherals attach to it.

- RC522 RFID Reader (SPI): Detects Mifare-standard cards up to 5 cm away and reads the 4-byte card UID. No card data is stored on the device.
- AS608 Fingerprint Sensor (UART at 57,600 baud): Stores pre-enrolled templates in onboard flash and runs matching locally. Only a confidence integer is returned. No fingerprint image leaves the sensor.
- OV2640 Camera Module: Captures a JPEG frame after fingerprint clearance and transmits it to the Python face verification service over HTTPS POST.
- SSD1306 OLED Display (I2C): Shows step-by-step verification status to the student, reducing uncertainty and time spent at the terminal.

## 3.3 Three-Factor Verification Flow

The terminal uses a strict sequential gate model. Each factor must clear its threshold before the next activates.

- 1) Student taps RFID card. Firmware reads the UID and sends an HTTPS GET to the backend to retrieve the student profile, including fingerprint template ID and stored face embedding.
- 2) Fingerprint sensor activates. Module compares against the linked template and returns a confidence score. Anything below 60 ends the session and logs a factor-level failure.
- 3) Camera fires one frame. The JPEG is base64-encoded and sent to the Python Flask service, which returns an accept or reject with a distance value.
- 4) If all three factors pass, firmware posts an attendance payload: student ID, terminal ID, UTC timestamp, and the three confidence values.
- 5) OLED shows the result. Backend writes to Firestore and emits a Socket.IO event. Open dashboard clients update within milliseconds.

## 3.4 Backend Layer

Firebase Firestore stores students, faculty, subjects, attendance records, marks, notifications, and roles. Firebase Authentication issues JWTs with custom claims encoding each user's role and linked entity ID. Firestore security rules enforce row-level access. Students read only their own records. Faculty write only to their assigned subjects. Parents read only their linked ward's subtree.

A Socket.IO server runs as a Node.js process. On receiving a valid attendance event, the server writes the Firestore document and emits a role-filtered Socket.IO event

immediately. Session context stores role and entity ID from the JWT, so each emit is scoped to the right audience without a per-event database lookup

### 3.5 Frontend Dashboards

Four React 19 modules share a common authentication context. React Router DOM v7 guards every dashboard route. Tailwind CSS handles styling. Recharts renders all charts. Framer Motion animates route transitions.

- Student Dashboard: attendance percentage per subject, session log, marks per assessment, semester trend graph, class attendance ranking, and a teacher notification feed.
- Faculty Dashboard: class attendance table with threshold alerts, marks entry form with per-assessment granularity, and a notification composer for individual parents or entire class groups.
- HOD Dashboard: faculty-wise summaries, department attendance trends, student performance distributions, notification log, and drill-down into individual profiles.
- Parent Dashboard: read-only. Subject-wise and session-level attendance, marks per subject, and a live notification feed from faculty.

## 4. IMPLEMENTATION

### 4.1 ESP32 Firmware

Firmware was written in the Arduino IDE using the ESP32 Arduino Core. The MFRC522 library handles SPI communication with the card reader. AS608 communication uses the Adafruit Fingerprint Sensor protocol over hardware UART2. ArduinoJSON builds and parses all JSON payloads. The main loop runs a state machine: idle state polls the RFID reader at 100 ms intervals, card detection transitions to verify state, and the machine returns to idle after each attempt regardless of outcome. All HTTPS calls use certificate pinning. Firmware updates deploy via Arduino OTA without physical terminal access.

### 4.2 Face Verification Service

The Python service is a Flask application with one endpoint: POST /verify. The request body carries a base64 JPEG. The service decodes it, calls `face_recognition.face_locations()` to find the largest face bounding box, then `face_recognition.face_encodings()` to compute the 128-dimensional embedding. It compares against the stored reference embedding using Euclidean distance.

Distance below 0.45 is a clean accept. Between 0.45 and 0.55 is flagged as low-confidence and logged for administrator review. Above 0.55 is rejected. We chose

this range by testing against 50 enrollment sets under three lighting conditions. Deploying this as a separate Flask microservice allows the recognition model to be replaced without touching any other component.

### 4.3 Firebase and Security

Firestore security rules are deployed via the Firebase CLI. The Socket.IO server decodes the Firebase JWT on connection and stores role and entity ID in session context. This controls which channels the client joins and which events are emitted throughout the session, avoiding per-event database lookups.

### 4.4 React Frontend

A context-based authentication provider decodes the Firebase JWT on login and exposes role and entity ID to all child components. Protected routes use a `PrivateRoute` wrapper that redirects unauthenticated users to the login screen. Each dashboard module is lazy-loaded to reduce initial bundle size. The Socket.IO client connects once after login. A `useReducer` hook manages attendance state, updating only affected components when a new event arrives. Recharts `ResponsiveContainer` ensures charts adapt to viewport changes without additional breakpoint logic.

## 5. RESULTS AND EVALUATION

### 5.1 Biometric Accuracy

We ran 312 verification attempts across 50 enrolled students. RFID read success was 100% within 0 to 5 cm. No misreads occurred at that distance.

The AS608 fingerprint module rejected valid fingers 2.6% of the time. All rejections occurred with students who had dry skin or a recent fingertip cut. False acceptances numbered zero across all 312 attempts. We re-enrolled the three highest-rejection students with improved placement guidance; their rejection rate dropped to zero in follow-up sessions.

Face verification using the `dlibResNet` model accepted valid faces at 96.4% under overhead fluorescent lighting, dropping to 90.5% under direct window backlight. Adding a positioning instruction to the OLED at the camera step recovered most of this gap in subsequent sessions. Combined across all three factors, 93.9% of attempts completed successfully.

### 5.2 End-to-End Latency

Latency was measured from RFID card tap to Socket.IO event received by an open dashboard client. Average end-

to-end latency was 6.7 seconds. Table 1 shows the breakdown by stage.

**Table 1:** End-to-End Pipeline Latency

Pipeline Stage	Avg (s)	Max (s)
RFID read and backend profile fetch	1.1	1.8
Fingerprint acquisition and on-module match	2.0	3.1
Frame capture and face verification	2.4	4.2
Firestore attendance record write	0.7	1.2
Socket.IO broadcast to dashboard clients	0.5	0.9
<b>Total (end-to-end)</b>	<b>6.7</b>	<b>11.2</b>

The face verification step dominates total latency. Moving inference on-device using TensorFlow Lite on an ESP32-S3 is the planned path to reduce this.

### 5.3 Dashboard Performance

Initial dashboard load from login to first data render averaged 1.3 seconds on campus Wi-Fi. Socket.IO events reached open clients within 480 milliseconds of each Firestore write on average. The slowest observed was 910 milliseconds during a session with eleven simultaneously logged-in clients. Faculty notifications reached parent dashboards within 3 seconds in all tests.

### 5.4 Comparative Analysis

Table 2 compares our system against prior single-factor approaches.

**Table 2:** Feature Comparison Against Prior Approaches

Feature	Prior Systems [1-7]	This Work
Identity check	Single factor only	RFID + fingerprint + face
Proxy attendance	Still possible	Eliminated by design
Hardware	GSM/standalone modules	ESP32 with onboard Wi-Fi
Live data sync	Absent	Firestore + Socket.IO
Dashboards	None or basic page	4 role-differentiated views
Parent notification	SMS only or absent	Real-time dashboard feed
Marks management	Not included	Integrated faculty interface
Peer ranking	Not available	Live class ranking

## 6. CONCLUSIONS

We set out to solve two problems that existing automated attendance systems leave open: weak identity confirmation at the point of capture, and the absence of a useful information layer for the stakeholders who depend on attendance data. Our system addresses both.

The hardware terminal enforces three sequential biometric checks before logging any event. Across 312 test attempts the combined success rate was 93.9% with zero false acceptances. The Firebase and Socket.IO backend delivered verified events to open dashboard clients in under 500 milliseconds on average. The four React 19 dashboards gave students, faculty, HODs, and parents direct access to the data they needed, without exposing data they should not see.

Three improvements are planned. First, Android and iOS companion apps using React Native so parents can receive notifications without a browser session. Second, a TensorFlow Lite model running on the ESP32-S3, removing the network-dependent Python service and making each terminal self-contained. Third, an attendance prediction module that flags students trending toward the 75% minimum threshold early enough for faculty to intervene.

## ACKNOWLEDGEMENT

We thank Mr. Lalit Saraswat, Guide and Assistant Professor, Department of Computer Science and Engineering, Raj Kumar Goel Institute of Technology, Ghaziabad, for his guidance, technical advice, and consistent support throughout this project.

## REFERENCES

- [1] I. Umar, A. Zulkifli, and N. A. Umar, "Student attendance management system using RFID and SMS," in Proc. ICEEOT, Chennai, India, 2016, pp. 468-472.
- [2] S. S. Patil, S. S. Chitnis, and A. M. Bagade, "Automatic attendance marking system using RFID," Int. J. Innovative Research in Computer and Communication Engineering, vol. 3, no. 4, pp. 3136-3142, Apr. 2015.
- [3] O. Shoewu and O. A. Idowu, "Development of attendance management system using biometrics," The Pacific Journal of Science and Technology, vol. 13, no. 1, pp. 300-307, May 2012.
- [4] I. Adjabi, A. Ouahabi, A. Benzaoui, and A. Taleb-Ahmed, "Past, present, and future of face recognition: A review," Electronics, vol. 9, no. 8, p. 1188, Aug. 2020.
- [5] G. Guo and N. Zhang, "A survey on deep learning based face recognition," Computer Vision and Image Understanding, vol. 189, p. 102805, 2019.

- [6] R. Bhatt and A. Bhatt, "A secure access control system using RFID and face recognition," in Proc. ICCCA, Greater Noida, India, 2017, pp. 1176-1180.
- [7] R. Sharma, A. Dhingra, and P. K. Gupta, "Smart attendance system using IoT and cloud computing," in Proc. IoT-SIU, Bhimtal, India, 2018, pp. 1-5.
- [8] R. Kumari, S. Gupta, and A. Khatri, "Web-based academic management system: A comprehensive review," Int. J. Engineering Research and Applications, vol. 6, no. 3, pp. 28-34, Mar. 2016.