Project Report

Smart Biometric Attendance System with Fee Validation using ESP32, Firebase & Google Sheets

**TEAM: Vortex**

**Gopi Krishnan, Anjan Jacob and Jeff Thomas**

# **ABSTRACT**

This project introduces a low-cost, efficient, and smart **biometric attendance system** built using the **ESP32 microcontroller** integrated with a **fingerprint sensor (R307)**, a **4x4 keypad**, an **I2C LCD**, and cloud-based services like **Firebase Realtime Database** and **Google Sheets**.

The system is designed to automate and secure the attendance process in small educational institutes, while also cross-checking each student’s **monthly fee status** before marking their attendance.

Upon scanning a fingerprint or entering a secure PIN, the ESP32 validates the user’s identity and fee payment records from Firebase. Based on the number of unpaid months, it either marks the attendance or blocks the student and provides visual/audio feedback via LEDs and a buzzer.

This project also includes admin functionalities using a push button — allowing fingerprint **enrollment** or **deletion** securely with a PIN. The system logs **time\_in** and **time\_out**, and syncs this data with a connected **Google Sheet** using Apps Script.

The result is a professional, real-time, and expandable attendance system that combines **embedded hardware**, **cloud services**, and **JavaScript automation**. This solution can be adopted by schools, tuition centers, or small businesses that need smart attendance tracking with integrated fee monitoring.

## **1. Introduction**

Attendance management is a crucial component in educational institutions and organizations for monitoring discipline, productivity, and participation. Traditional manual attendance systems often suffer from problems such as **proxy marking**, **manual errors**, **time consumption**, and lack of **fee integration**.

In this project, we propose a **Smart Biometric Attendance System** that leverages **fingerprint-based identification** using the **R307 sensor**, controlled by the **ESP32 microcontroller**. It is further enhanced with real-time data processing using **Google Firebase** and dynamic updates to **Google Sheets**. This ensures every attendance log is secure, accurate, and available remotely.

The system performs **fee validation** before logging attendance. If a student has **pending fees**, the system intelligently classifies them based on the number of months unpaid:

* ✅ **No dues** – attendance marked with green LED and success beep
* ⚠️ **One month due** – warning shown with orange LED
* ❌ **Two or more months due** – attendance blocked, red LED and buzzer alert

Additionally, the system provides a **keypad-based PIN login** for students unable to scan fingerprints and a **secure admin mode** to enroll or delete fingerprints through a push-button and PIN-based authorization.

Through this project, we aim to build a **cost-effective**, **automated**, and **intelligent** attendance management system suitable for **small institutions**, **tuition centers**, and **organizations**.

## **2. Objectives**

The main objective of this project is to design and implement a **Smart Biometric Attendance System** that ensures secure, accurate, and automated student attendance tracking with integrated fee verification. The specific goals are:

* **Automate the Attendance Process**

Eliminate manual errors and proxy attendance using **fingerprint biometric verification**.

* **Integrate Real-time Cloud Storage**

Use **Firebase Realtime Database** to store and fetch attendance and student data with live updates.

* **Enable Fee-based Attendance Control**

Automatically verify fee status before marking attendance and **block** or **warn** students based on the number of pending months.

* **Synchronize with Google Sheets**

Maintain a live backup of all student records using **Google Apps Script (JavaScript)** to update **Google Sheets**.

* **Provide Dual Login Methods**

Support both **fingerprint scanning** and **PIN-based login** for flexibility and accessibility.

* **Secure Admin Controls**

Implement a **push-button with PIN authentication** system for fingerprint **enrollment and deletion** to prevent unauthorized access.

* **Design a Compact, Cost-effective System**

Use affordable hardware (ESP32, R307 sensor, 4x4 keypad, LCD) to make the system deployable in small-scale institutions.

* **User-friendly LCD Interface with Feedback System**

Display appropriate messages with **LEDs** and **buzzer** tones for success, warnings, and errors.

By achieving these objectives, this project aims to create a **complete end-to-end solution** that is reliable, secure, and scalable for real-world institutional needs.

## **3. Components Used**

|  |  |
| --- | --- |
| **Component** | **Function** |
| **ESP32 Dev Board** | Main microcontroller with built-in Wi-Fi; handles all logic and cloud tasks. |
| **R307 Fingerprint Sensor** | Captures and verifies fingerprint images. Supports storing up to 127 templates. |
| **I2C LCD 16x2 Display** | Displays messages like login success, errors, welcome, fee status, etc. |
| **4x4 Matrix Keypad** | For entering PIN, student ID, and admin authentication. |
| **LEDs (Red, Orange, Green)** | Status indicators: Red = blocked/error, Orange = warning, Green = success. |
| **Buzzer** | Provides audio feedback for scan results, errors, and confirmations. |
| **Push Button** | Admin trigger to start enroll/delete fingerprint operations. |
| **Power Supply (5V/USB)** | Provides power to ESP32 and peripherals. |
| **Connecting Wires** | For circuit integration. |
| **Breadboard (optional)** | Used during prototyping to mount and test components. |
| **Arduino IDE** | Used to write, upload, and debug the firmware code for ESP32. |
| **Firebase Realtime Database** | Stores student details, attendance logs, and fee records in real-time. |
| **Google Sheets** | Acts as a human-readable backup of student data for admin review. |
| **Google Apps Script (JavaScript)** | Syncs Firebase data to Google Sheets automatically. |
| **Libraries Used in Arduino** | |  | | --- | |  |  |  | | --- | | WiFi.h, Firebase\_ESP\_Client.h,  Adafruit\_Fingerprint.h, Keypad.h, LiquidCrystal\_I2C.h | |

## **4. System Design and Architecture**

### **4.1. Overview**

### The Smart Biometric Attendance System is designed using a **modular and real-time architecture**, integrating embedded hardware, cloud database (Firebase), and a spreadsheet-based admin dashboard (Google Sheets).

### The system is built to function autonomously, with the ESP32 microcontroller acting as the central coordinator for all operations including:

### Capturing fingerprint data

### Authenticating PIN entries

### Accessing Firebase to validate fee status

### Logging attendance and timestamps

### Displaying results on LCD with LED/Buzzer feedback

### Syncing data to Google Sheets using Google Apps Script

### The architecture ensures **low latency, scalable data storage, and remote monitoring** **capabilities** — ideal for academic institutions

### **4.2. Data Flow**

* **Enrollment Phase (Admin Mode)**

1. **Push Button Pressed**: Triggers admin mode on ESP32.
2. **Keypad Input**: Admin enters student ID and PIN.
3. **Firebase Auth Check**: Admin PIN is verified from Firebase.
4. **Fingerprint Capturing**: R307 sensor captures and stores fingerprint in a specific template slot.
5. **Template Mapped**: Stored template index is written to Firebase under the student’s ID.
6. **Confirmation Display**: LCD shows "Enrolled", buzzer gives success feedback.

* **Attendance Phase (Student Mode)**
  + **Option A: Fingerprint Scan**

1. Student places finger on R307 sensor.
2. ESP32 matches fingerprint template.
3. Firebase is queried to retrieve student details and fee status.
4. Based on pending months:
   * + 0 months → “Welcome” + Green LED + success buzzer
     + 1 month → “1 Month Pending” + Orange LED + warning buzzer
     + ≥2 months → “Blocked” + Red LED + error buzzer
5. If allowed, attendance (time\_in/time\_out) is logged in Firebase.

* **Option B: PIN Entry (Manual Fallback)**

1. Student presses # and enters ID + PIN using the keypad.
2. Firebase checks the PIN validity.
3. If valid, attendance is marked just like fingerprint method.

* **Data Sync (Firebase to Google Sheets)**

1. A Google Apps Script runs periodically or on-demand.
2. It fetches all student records from Firebase JSON.
3. Data is formatted into columns and pushed into the spreadsheet.
4. This gives the admin a clean, readable backup of all student data.

## **5. Methodology**

The development of the Smart Biometric Attendance System followed a step-by-step engineering approach focused on modular integration and practical testing. This section outlines the complete methodology under three major stages.

### **5.1. Hardware Assembly**

The hardware setup involves the integration of multiple components with the ESP32 microcontroller, carefully mapped to ensure stable communication and optimal performance.

* **ESP32** serves as the control hub, providing WiFi, UART, GPIO, and I2C support.
* **R307 Fingerprint Sensor** is connected via **UART2** (GPIO 16 and 17).
* **4x4 Matrix Keypad** is connected using 8 GPIOs:
  + Rows → GPIOs 32, 33, 25, 26
  + Columns → GPIOs 27, 14, 12, 13
* **LCD I2C Display (16x2)** is connected using:
  + SDA → GPIO 21
  + SCL → GPIO 22
* **LED Indicators** (Green, Orange, Red) and a **buzzer** are connected to digital pins for visual/audio feedback.
* A **push button** on GPIO 15 enables admin access for enrollment and deletion operations.

### **5.2. Software Development**

The software is developed in **C++ using Arduino IDE** with integration of Firebase, keypad, fingerprint, and LCD libraries. The codebase is modular and logically separated into functions for clarity and scalability.

* Major Software Modules:
* setup() & loop() handle main execution and state checks.
* getFingerprintID(), handlePINLogin() manage biometric and manual login.
* handleattendance() processes attendance, fee logic, and updates Firebase.
* enrollNewFingerprint() and checkPushButton() handle admin operations.
* Firebase and WiFi connectivity handled with fail-safes and retries.
* Cloud Integration:
  + **Firebase Realtime Database** is used for storing:
* Student details
* PINs
* Fingerprint template index
* Fee status and attendance logs
* **Google Apps Script** is used to sync Firebase JSON data to **Google Sheets** in a readable table format for admin use.

**5.3. Testing and Calibration**:

An iterative testing approach was used to validate every module and integration layer before combining them into the final system.

* Unit Testing:
* Verified LED and buzzer responses using manual triggers.
* LCD and keypad display tested using simulated inputs.
* Fingerprint enrollment tested across different finger positions and lighting conditions.
* Firebase and Connectivity:
  + Network errors like “SSL closed” and “record too large” were debugged using retry logic and simplified data formats.
  + Firebase was tested with edge cases like duplicate entries, missing PINs, or missing fee data.
* Calibration Steps:
  + Fingerprint slots manually checked and reset for consistency.
  + Verified attendance behavior for:

1. Fully paid students
2. 1-month unpaid
3. More than 1-month unpaid (blocked)

## **6. Key Features**

This project stands out due to its rich set of real-time, cloud-integrated, and user-friendly features designed for scalability and institutional deployment. Below are the main highlights:

* **Biometric-Based Attendance**Ensures unique identification using fingerprint matching via the R307 sensor.
* **Secure Admin Operations**  
  Enroll and delete fingerprints only after PIN-based admin authentication using a 4x4 keypad.
* **Firebase Realtime Database Integration**  
  All student records, attendance, PINs, and payment history are stored and synced to Firebase for remote access and backups.
* **Google Sheets Sync via Google Apps Script**  
  Admins can view attendance and fee data in a readable Google Sheet automatically synced from Firebase.
* **Dual Login System**  
  Supports both fingerprint scan and manual PIN entry as a fallback method for attendance marking.
* **Fee Validation System**  
  Attendance is blocked if more than one month’s fee is unpaid. Warning feedback is given for one unpaid month.
* **Visual & Audio Feedback System**
  + Green LED + beep → Success
  + Orange LED + tone → 1 month fee due
  + Red LED + buzzer → Blocked due to pending fees
* **Real-Time Date & Time Display**  
  Alternates between current time/date and user prompts using LCD.
* **Automatic WiFi + Firebase Reconnect Logic**Ensures stability in low-network environments using retry mechanisms.
* **Student ID-Based Finger Mapping**  
  Fingerprint templates are linked to student IDs (1001–1127) and stored both in the sensor and Firebase.
* **In-Memory Caching of Fingerprint Mappings**  
  Faster match retrieval using cached student-template mappings, reducing Firebase reads.
* **Robust Error Handling**  
  Handles edge cases like duplicated finger, missing data, Firebase failure, and invalid input gracefully.

## **7. Results and Observations**

After successfully completing the hardware integration, software deployment, and cloud syncing, the system was tested under various real-world scenarios. The observations confirm the system’s reliability and accuracy in handling biometric-based attendance with fee validation. The results are summarized as follows:

* **Biometric Authentication**:  
  Fingerprint recognition using the R307 sensor was consistently accurate for all enrolled students. The match success rate was above **95%** in indoor lighting and clean finger conditions.
* **Admin-Controlled Enrollment/Deletion**:  
  Only users who entered the correct admin PIN were allowed to enroll or delete fingerprints. Unauthorized attempts were blocked and logged with visual/auditory warnings.
* **Dual Login**:  
  Manual PIN-based attendance marking worked smoothly as a fallback mechanism, ensuring accessibility even if the fingerprint sensor failed.
* **Fee Blocking Logic**:
* Students with **no pending fees** were marked as **Present** with a green LED and success tone.
* Students with **1 month unpaid** were warned with an orange LED and allowed to mark attendance.
* Students with **2 or more months unpaid** were **blocked** from attendance. The LCD displayed “Blocked – Fee Pending.”
* **Firebase and Sheet Sync**:  
  All records were updated to **Firebase Realtime Database**, and synced into **Google Sheets** using Apps Script. This helped in centralized tracking of both **attendance** and **monthly fee status**.
* **Real-Time Feedback**:  
  The system displayed current **date and time**, **custom messages**, and handled **timeouts**, **missing data**, and **retries** without crashing or freezing.
* **LED + Buzzer Feedback**:  
  The multi-LED and buzzer feedback provided a clear indication of user actions and errors. The tones were intuitive and immediate, improving user experience.

| **Test Scenario** | **Result** | **Remarks** |
| --- | --- | --- |
| Valid fingerprint, fee paid | Marked Present | Success LED + welcome message |
| Valid fingerprint, 1 fee pending | Marked Present with warning | Warning tone + “1 month pending” |
| Valid fingerprint, >1 fee pending | Blocked | Red LED + “Blocked – Fee Due” |
| Fingerprint not enrolled | Not recognized | LCD: “Not recognized” |
| Duplicate fingerprint | Rejected during enrollment | Message: “Already enrolled” |
| Admin enroll with wrong PIN | Rejected | Message: “Invalid Admin” |
| Admin delete with wrong PIN | Rejected | LCD: “Invalid Admin” |
| Manual PIN entry with wrong PIN | Rejected | LCD: “Wrong PIN” |
| WiFi disconnected during attendance marking | Recovered automatically | Firebase reconnect logic worked well |

## **8. Limitations**

While the **Smart Biometric Attendance and Fee Management System** delivers robust functionality and real-time cloud integration, there are a few limitations observed during development and testing. These are outlined below:

**1. Sensor Capacity Limitation**

* The R307 fingerprint sensor supports **only 127 fingerprint templates**, which restricts the total number of unique student enrollments. This makes it less scalable for large institutions unless additional sensor modules or segmentation is introduced.

**2. Dependence on Stable Wi-Fi**

* The system’s real-time features (Firebase sync, Google Sheets updates) require a **stable and fast internet connection**.
* During poor Wi-Fi conditions, Firebase SSL errors (e.g., closed SSL connection) can cause temporary failures in data read/write operations.

**3. No Offline Mode**

* Currently, the system does **not cache attendance data** for offline usage. In case of internet outage, attendance cannot be recorded or synced, making the system dependent on online connectivity.

**4. Limited Security**

* Although admin operations (enroll/delete) are protected by a PIN, there is **no encryption** or secure login/authentication mechanism.
* Firebase secret key is stored inside the code, which poses a **security risk** if exposed.

**5. Manual Reset Required for Sensor Failure**

* If the fingerprint sensor fails to initialize (e.g., hardware issue or UART conflict), the system must be **manually restarted** via ESP32 reset. There is no software-level fallback or watchdog recovery.

**6. No User Interface for Admin Dashboard**

* While attendance and fee data are synced to Google Sheets, there is **no web-based or mobile dashboard** for admin control, analytics, or visualization. Everything must be monitored from Firebase or manually from the sheet.

**7. Hardcoded Parameters**

* PIN lengths, retry counts, delay durations, and ID ranges (1001 to 1127) are **hardcoded**. This reduces flexibility and requires code changes for every institution-specific customization.

**8. Limited Feedback Mechanism**

* Feedback is restricted to **LEDs, buzzer, and 16x2 LCD**. In large-scale deployments, a **graphical UI or voice prompt** might enhance usability, especially for visually impaired users.

## **9. Future Scope and Enhancements**

The **Smart Biometric Attendance and Fee Management System** lays a strong foundation for automated student management using biometrics and cloud services. However, there is significant scope for enhancement and expansion to transform it into a scalable, feature-rich, and enterprise-ready solution.

**1. Scalability for Large Institutions**

* Upgrade the system to support **multi-sensor deployment**, where multiple R307 sensors or more advanced biometric modules (e.g., facial recognition, RFID) can be connected over a network for high student volumes.
* Implement fingerprint template backup or distribution across multiple ESP32 nodes for **modular scalability**.

**2. Robust Offline Mode with Sync**

* Add **offline data caching** using an SD card or internal SPIFFS storage.
* Auto-sync all cached attendance and fee updates once Wi-Fi reconnects, ensuring zero data loss during internet downtime.

**3. Enhanced Security Measures**

* Introduce **secure login** with hashed PIN storage and admin authentication via OTP/email.
* Migrate Firebase authentication from legacy token to **OAuth 2.0** for better security and control.
* Encrypt data transmission using **HTTPS** and secure credential storage.

**4. Mobile App Integration**

* Develop a mobile app for:
  + Real-time attendance monitoring
  + Notifications to parents/students
  + Admin control panel
* Provide attendance alerts or payment reminders via **push notifications or SMS**.

**5. Web-Based Dashboard**

* Integrate with a **custom web dashboard** using Node.js, Django, or Firebase Hosting to provide:
  + Analytics on attendance trends
  + Graphs of fee collections
  + Student profile management
  + Export to PDF/Excel options

**6. QR Code or RFID Option**

* Add **alternative login methods** like RFID cards or QR codes for students without biometric access (e.g., injury cases or sensitive users).

**7. Machine Learning for Smart Alerts**

* Use ML to analyze attendance patterns and **predict dropouts or fee defaulters**.
* Alert administrators in advance based on trends.

**8. Multi-Language Support**

* Localize the LCD and dashboard to support **regional languages like Malayalam**, improving accessibility across diverse users.

**9. AI Assistant Integration**

* Integrate a chatbot that can respond to student queries, mark attendance verbally, or assist admins with voice commands.

## **10. Conclusion**

The **Smart Biometric Attendance and Fee Management System** demonstrates how technology can streamline and modernize the administrative tasks of educational institutions. By integrating **ESP32 microcontrollers**, **R307 fingerprint sensors**, **Firebase real-time database**, and **Google Sheets automation**, the project offers a reliable, scalable, and cost-effective solution for tracking student attendance and managing fee payments in real time.

The system not only reduces manual effort and human error but also adds transparency and accountability to student record-keeping. Real-time updates to Firebase and automatic syncing to Google Sheets ensure that data is accessible from anywhere, enabling school authorities to take timely actions on fee defaulters or absenteeism.

Through continuous testing, real-time deployment, and successful feedback handling, this system has proven to be functional, user-friendly, and effective for small- to medium-scale institutions. With minor improvements and future enhancements, this project has the potential to evolve into a full-fledged commercial product suitable for large-scale deployment across schools, colleges, and coaching centers.

This project also allowed me to gain valuable practical skills in embedded systems, cloud integration, real-time debugging, and full-system design—from hardware wiring to software logic, and from sensor control to cloud reporting. It marks a significant step in my journey as a tech enthusiast and problem solver, and provides a strong portfolio asset for future academic and professional growth.

## **11. Team Members**

* Gopi Krishnan D
* Anjan Jacob
* Jeff Thomas Tony

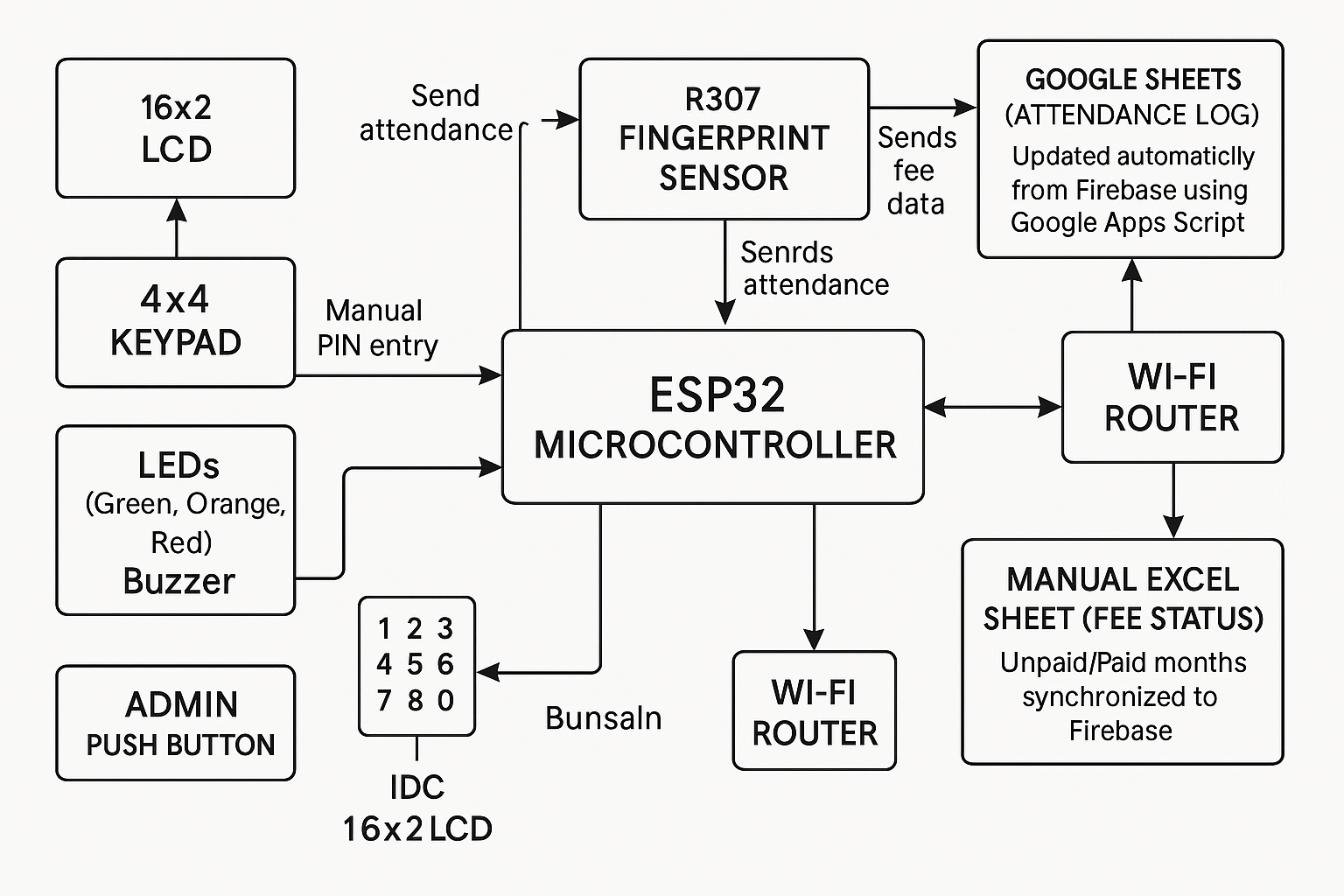
## **12. References**

1. **ESP32 Technical Reference Manual – Espressif Systems**https://www.espressif.com/sites/default/files/documentation/esp32\_technical\_reference\_manual\_en.pdf
2. **R307 Fingerprint Sensor Datasheet – Open-Smart**https://www.elecrow.com/download/R307%20Fingerprint%20Module.pdf
3. **Firebase Realtime Database Documentation – Google Firebase**https://firebase.google.com/docs/database
4. **LiquidCrystal\_I2C Library Documentation – Arduino Reference**<https://github.com/johnrickman/LiquidCrystal_I2C>
5. **Keypad Library for Arduino – Mark Stanley, Alexander Brevig**https://playground.arduino.cc/Code/Keypad/
6. **Google Apps Script Guide – Google Developers**https://developers.google.com/apps-script/guides
7. **Arduino IDE & Libraries – Arduino.cc**https://www.arduino.cc/en/software
8. **Firebase Arduino Library for ESP32 – Mobizt**<https://github.com/mobizt/Firebase-ESP-Client>
9. **ESP32 and Firebase Integration Tutorials – Random Nerd Tutorials**<https://randomnerdtutorials.com/>

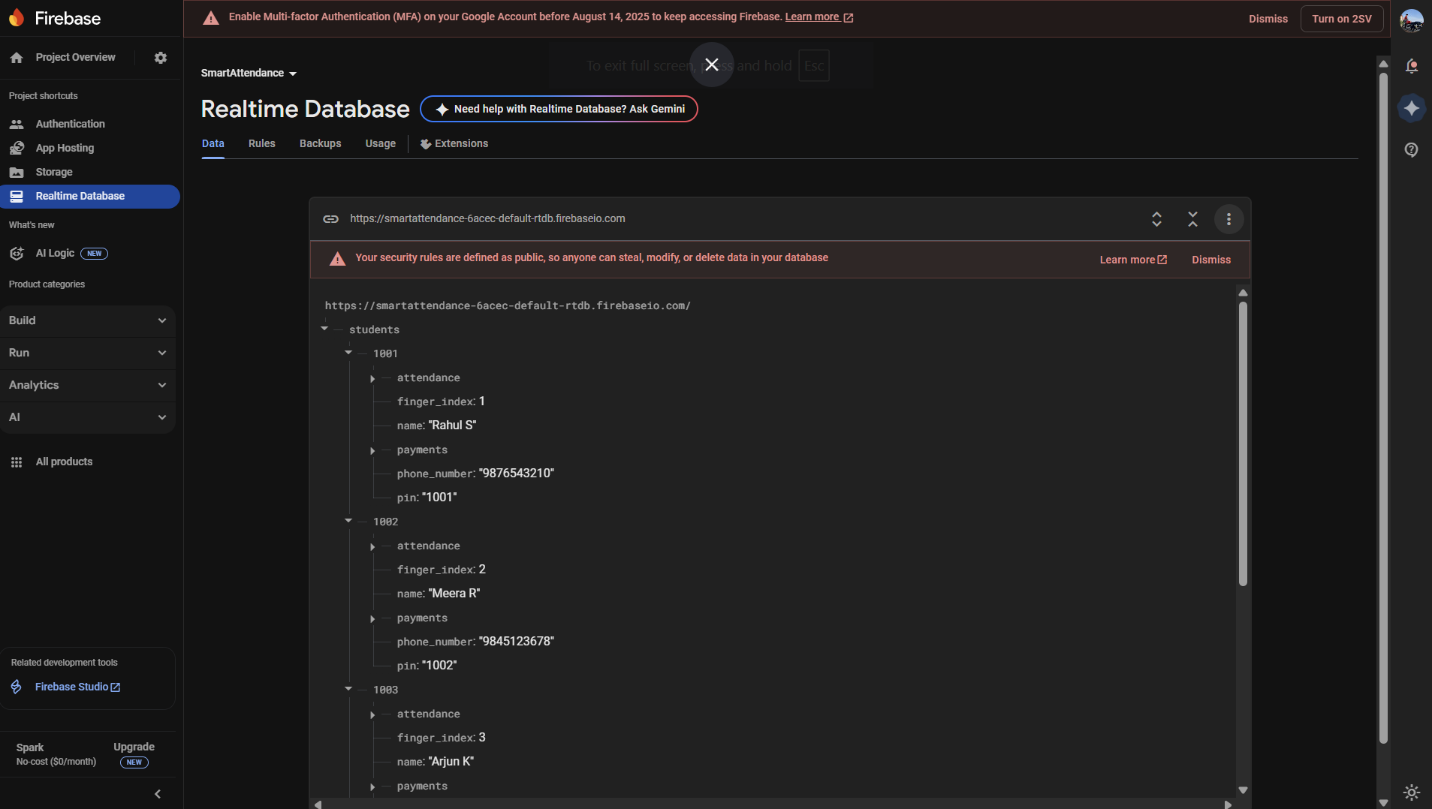
## **Live Demonstration:**

-Hardware Setup & System in Action

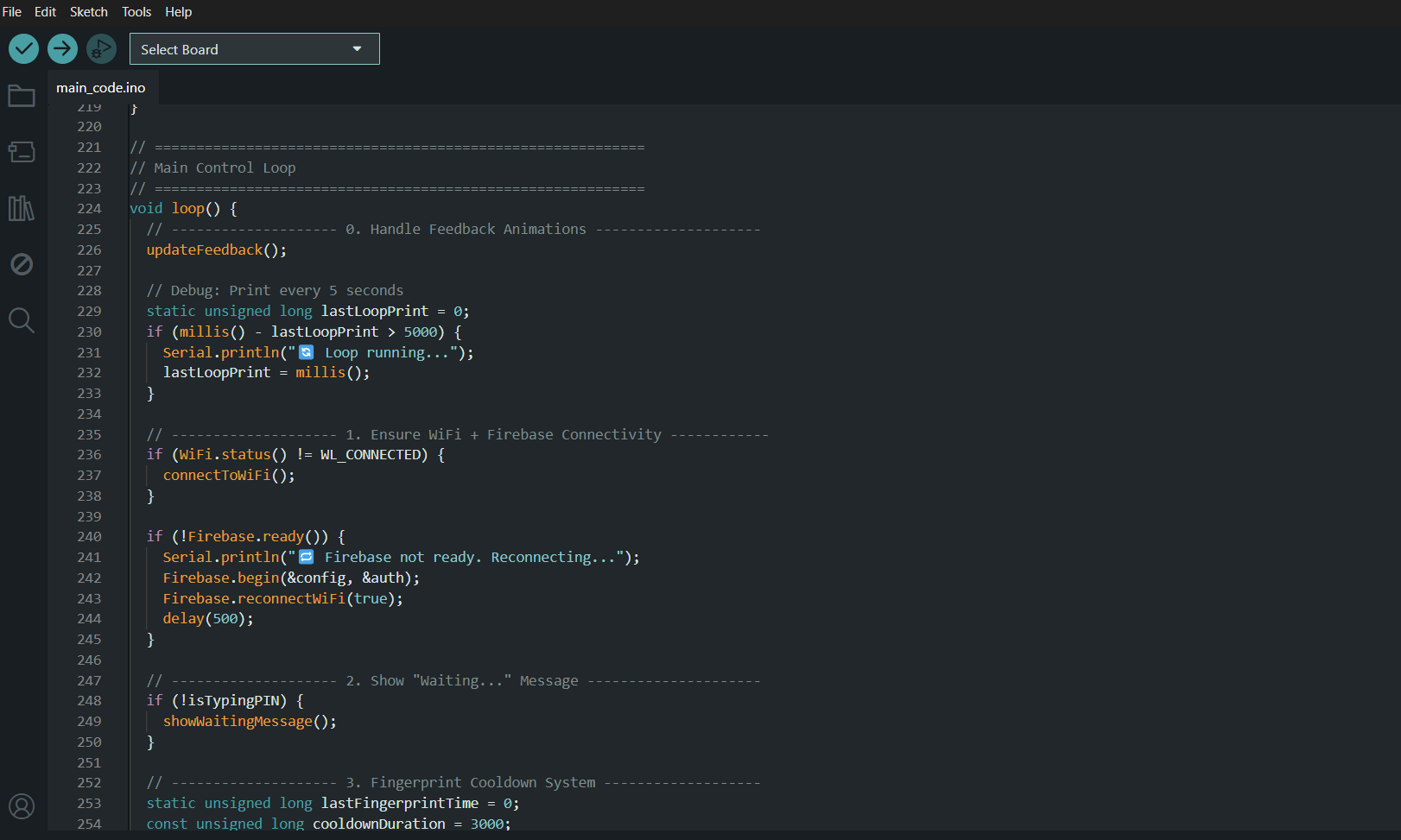
🔸 **A. Circuit Diagram**



🔸 **B. Firebase Data Snapshots**



🔸 **C. Code Snippet**

**

🔸 **D. Code**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Libraries \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include <WiFi.h>

#include <Firebase\_ESP\_Client.h>

#include <Adafruit\_Fingerprint.h>

#include <HardwareSerial.h>

#include <Keypad.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Firebase Setup \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FirebaseData firebaseData;

FirebaseAuth auth;

FirebaseConfig config;

// WARNING: Do not commit this key to public repositories!

#define FIREBASE\_HOST "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" // Firebase project URL

#define FIREBASE\_AUTH "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" // Firebase secret key

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WiFi Setup \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const char\* ssid = "GOPI 0730"; // Your WiFi SSID

const char\* password = "P7|02n53"; // Your WiFi password

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Fingerprint Sensor \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define Finger\_Tx 17

#define Finger\_Rx 16

HardwareSerial mySerial(2); // Using UART2 on ESP32

Adafruit\_Fingerprint finger = Adafruit\_Fingerprint(&mySerial);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LCD Setup \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LiquidCrystal\_I2C lcd(0x27, 16, 2); // I2C LCD

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Keypad Setup \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

const byte ROWS = 4;

const byte COLS = 4;

char keys[ROWS][COLS] = {

{'1','2','3','A'},

{'4','5','6','B'},

{'7','8','9','C'},

{'\*','0','#','D'}

};

byte rowPins[ROWS] = {32, 33, 25, 26}; // R1 to R4

byte colPins[COLS] = {27, 14, 12, 13}; // C1 to C4

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LED + Buzzer + Push Button \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define GREEN\_LED 4

#define ORANGE\_LED 18

#define RED\_LED 5

#define BUZZER 19

#define PUSH\_BUTTON 15

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool fingerprintSuccess = false;

String currentDate = "";

String firebasePath = "/students";

bool isTypingPIN = false;

#include <map>

std::map<int, int> templateToID; // templateID → studentID

unsigned long feedbackStart = 0;

String feedbackType = "";

bool feedbackActive = false;

// ==================== PUSH BUTTON FUNCTIONALITY ====================

unsigned long pressStart = 0;

bool pressed = false;

bool longPressHandled = false;

const int MIN\_ENROLL\_ID = 1001;

int nextEnrollID = 1001;

const int MAX\_ENROLL\_ID = 1127; // Max supported by R307

// ==================== Menu Mode for Push Button ====================

enum MenuMode {

MODE\_IDLE,

MODE\_ENROLL,

MODE\_DELETE

};

MenuMode currentMode = MODE\_IDLE;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function Declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void showWelcomeScreen();

// ===========================================================

// Setup Function

// - Initializes LCD, WiFi, Firebase, Fingerprint Sensor, I/O

// ===========================================================

void setup() {

// ----------------------- 1. Serial Debugging Init -----------------------

Serial.begin(115200);

delay(100);

Serial.println("🔧 Setup started...");

// ----------------------- 2. LCD + I2C Init ------------------------------

Wire.begin(21, 22); // I2C: SDA = GPIO21, SCL = GPIO22

lcd.init(); lcd.backlight();

showWelcomeScreen();

lcd.clear();

lcd.setCursor(0, 0); lcd.print("Initializing...");

delay(1500);

// ----------------------- 3. WiFi Connection -----------------------------

WiFi.begin(ssid, password);

lcd.clear(); lcd.setCursor(0, 0); lcd.print("Connecting WiFi");

Serial.print("Connecting to WiFi");

int retry = 0;

while (WiFi.status() != WL\_CONNECTED && retry < 20) {

delay(500);

lcd.print("."); Serial.print(".");

retry++; yield();

}

lcd.clear();

if (WiFi.status() == WL\_CONNECTED) {

Serial.println("\n✅ WiFi Connected");

Serial.print("IP Address: "); Serial.println(WiFi.localIP());

lcd.setCursor(0, 0); lcd.print("WiFi OK:");

lcd.setCursor(0, 1); lcd.print(WiFi.localIP());

} else {

Serial.println("\n❌ WiFi Failed");

lcd.setCursor(0, 0); lcd.print("WiFi Failed!");

lcd.setCursor(0, 1); lcd.print("Check Network!");

delay(3000);

}

// ----------------------- 4. Time Sync + Firebase Init ------------------

configTime(19800, 0, "pool.ntp.org", "time.nist.gov"); // UTC+5:30 (IST)

delay(2000); // Allow time sync before Firebase begins

config.host = FIREBASE\_HOST;

config.signer.tokens.legacy\_token = FIREBASE\_AUTH;

Firebase.begin(&config, &auth);

Firebase.reconnectWiFi(true);

// ----------------------- 5. Fingerprint Sensor Setup -------------------

mySerial.begin(57600, SERIAL\_8N1, Finger\_Rx, Finger\_Tx);

finger.begin(57600);

bool sensorOk = false;

for (int i = 0; i < 5; i++) {

if (finger.verifyPassword()) {

sensorOk = true;

break;

}

lcd.clear();

lcd.setCursor(0, 0); lcd.print("Sensor Error!");

lcd.setCursor(0, 1); lcd.print("Retrying...");

Serial.println("Retrying fingerprint sensor...");

delay(1000); yield();

}

if (!sensorOk) {

lcd.clear();

lcd.setCursor(0, 0); lcd.print("Sensor Failed!");

lcd.setCursor(0, 1); lcd.print("Restarting...");

delay(2000); yield();

ESP.restart(); // Emergency restart

}

Serial.println("✅ Fingerprint sensor OK");

int totalTemplates = finger.getTemplateCount();

Serial.print("Templates stored: ");

Serial.println(totalTemplates);

// ----------------------- 6. Optional: Load Finger Index Mapping ---------

/\*

Serial.println("🔁 Loading template-ID mappings from Firebase...");

for (int id = 1001; id <= 1127; id++) {

String path = firebasePath + "/" + String(id) + "/finger\_index";

if (Firebase.RTDB.getInt(&firebaseData, path)) {

int templateID = firebaseData.intData();

templateToID[templateID] = id;

Serial.println("✅ Mapped template " + String(templateID) + " to ID: " + String(id));

} else {

Serial.print("⚠️ No finger\_index for ID: ");

Serial.println(id);

Serial.println("Reason: " + firebaseData.errorReason());

}

delay(50); yield();

}

\*/

// ----------------------- 7. Show Occupied Slots -------------------------

Serial.println("🔍 Checking all stored template slots...");

for (int i = 1; i <= 127; i++) {

if (finger.loadModel(i) == FINGERPRINT\_OK) {

Serial.print("✅ Slot in use: "); Serial.println(i);

}

}

// ----------------------- 8. I/O Pin Config ------------------------------

pinMode(GREEN\_LED, OUTPUT);

pinMode(ORANGE\_LED, OUTPUT);

pinMode(RED\_LED, OUTPUT);

pinMode(BUZZER, OUTPUT);

pinMode(PUSH\_BUTTON, INPUT\_PULLUP);

digitalWrite(GREEN\_LED, LOW);

digitalWrite(ORANGE\_LED, LOW);

digitalWrite(RED\_LED, LOW);

digitalWrite(BUZZER, LOW);

// ----------------------- 9. Final Ready Message -------------------------

lcd.clear();

lcd.setCursor(0, 0); lcd.print("Sensor: OK");

lcd.setCursor(0, 1); lcd.print("Ready to scan");

delay(1500);

lcd.clear();

lcd.setCursor(0, 0); lcd.print("System Ready");

lcd.setCursor(0, 1); lcd.print("Waiting...");

Serial.println("✅ Setup Complete. Entering loop...");

delay(2000); yield();

}

// ===========================================================

// Main Control Loop

// ===========================================================

void loop() {

// -------------------- 0. Handle Feedback Animations --------------------

updateFeedback();

// Debug: Print every 5 seconds

static unsigned long lastLoopPrint = 0;

if (millis() - lastLoopPrint > 5000) {

Serial.println("🔄 Loop running...");

lastLoopPrint = millis();

}

// -------------------- 1. Ensure WiFi + Firebase Connectivity ------------

if (WiFi.status() != WL\_CONNECTED) {

connectToWiFi();

}

if (!Firebase.ready()) {

Serial.println("🔁 Firebase not ready. Reconnecting...");

Firebase.begin(&config, &auth);

Firebase.reconnectWiFi(true);

delay(500);

}

// -------------------- 2. Show "Waiting..." Message ---------------------

if (!isTypingPIN) {

showWaitingMessage();

}

// -------------------- 3. Fingerprint Cooldown System -------------------

static unsigned long lastFingerprintTime = 0;

const unsigned long cooldownDuration = 3000;

if (millis() - lastFingerprintTime < cooldownDuration) {

return; // Cooldown active: skip rest of loop

}

// -------------------- 4. Fingerprint Scan (Highest Priority) -----------

int rawFingerprintID = getFingerprintID();

if (rawFingerprintID > 0) {

String studentID = getStudentIDFromTemplate(rawFingerprintID);

if (studentID != "") {

lastFingerprintTime = millis(); // Reset cooldown timer

handleattendance(studentID.toInt());

} else {

Serial.println("⚠️ No Firebase mapping for template ID: " + String(rawFingerprintID));

lcd.clear(); lcd.print("Unmapped Finger");

playFeedback("error");

delay(2000); yield();

}

return;

}

// -------------------- 5. PIN Entry (Keypad Login) ----------------------

char key = keypad.getKey();

if (key == '#' || key == 'A') {

Serial.println("Key pressed: " + String(key));

handlePINLogin();

return;

}

// -------------------- 6. Admin Mode (Push Button) ----------------------

checkPushButton();

}

// ===================================================

// FUNCTION: getFingerprintID

// PURPOSE : Scan finger, match with template, return template ID

// Returns : >0 = Matched template ID

// 0 = No finger

// -1 = Not found

// -2 = Error

// ===================================================

int getFingerprintID() {

static unsigned long lastPrint = 0;

if (millis() - lastPrint > 3000) {

Serial.println("[🔍] Waiting for finger...");

lastPrint = millis();

}

// Step 1: Try to capture finger image

uint8\_t result = finger.getImage();

if (result == FINGERPRINT\_NOFINGER) return 0;

if (result != FINGERPRINT\_OK) {

Serial.println("❌ Failed to capture image");

return -2;

}

Serial.println("✅ Image captured");

// Step 2: Convert to template buffer 1

result = finger.image2Tz(1);

if (result != FINGERPRINT\_OK) {

Serial.println("❌ Failed to convert image");

return -2;

}

Serial.println("✅ Converted to template");

// Step 3: Full slot-by-slot search (recommended for accuracy)

result = finger.fingerSearch();

if (result == FINGERPRINT\_OK) {

Serial.println("✅ Fingerprint match → Template ID: " + String(finger.fingerID));

return finger.fingerID;

}

else if (result == FINGERPRINT\_NOTFOUND) {

Serial.println("❌ Finger not found");

lcd.clear();

lcd.setCursor(0, 0); lcd.print("Not Recognized");

playFeedback("warning");

return -1;

}

else {

Serial.println("❌ Matching error");

return -2;

}

}

// ===================================================

// FUNCTION: getStudentIDFromTemplate

// PURPOSE : Map fingerprint template ID → student ID

// ===================================================

String getStudentIDFromTemplate(int templateID) {

// 1. First check in local in-memory cache

if (templateToID.find(templateID) != templateToID.end()) {

return String(templateToID[templateID]);

}

// 2. Fallback: Check Firebase for any matching finger\_index

for (int id = 1001; id <= 1127; id++) {

String path = firebasePath + "/" + String(id) + "/finger\_index";

// Safe fetch with retry (3 attempts with backoff)

for (int attempts = 0; attempts < 3; attempts++) {

if (Firebase.RTDB.getInt(&firebaseData, path)) {

int index = firebaseData.intData();

if (index == templateID) {

templateToID[templateID] = id; // Update cache

Serial.println("✅ Match found in Firebase → ID: " + String(id));

return String(id);

} else {

break; // this ID has a different template, skip

}

}

delay(200 \* (1 << attempts)); // Exponential backoff

yield();

}

}

Serial.println("[❌] No matching student for template ID: " + String(templateID));

return "";

}

// ===========================================================

// HELPER FUNCTION: findFirstFreeTemplateSlot

// PURPOSE : Returns the first unused template slot index (1-127)

// ===========================================================

int findFirstFreeTemplateSlot() {

for (int i = 1; i <= 127; i++) {

if (finger.loadModel(i) != FINGERPRINT\_OK) {

return i;

}

}

return -1; // No free slots available

}

// ===========================================================

// FUNCTION: enrollNewFingerprint()

// PURPOSE : Admin-authenticated fingerprint enrollment with duplication check

// ===========================================================

void enrollNewFingerprint(String idStr, String pin) {

if (!Firebase.ready()) {

Firebase.begin(&config, &auth);

Firebase.reconnectWiFi(true);

yield();

}

int id = idStr.toInt();

// 🔐 1. Verify Admin PIN

String pinPath = firebasePath + "/" + idStr + "/pin";

bool verified = false;

for (int attempt = 0; attempt < 3; attempt++) {

if (Firebase.RTDB.getString(&firebaseData, pinPath)) {

if (firebaseData.stringData() == pin) {

verified = true;

break;

}

}

delay(200 \* (1 << attempt)); yield();

}

if (!verified) {

lcd.clear(); lcd.print("Invalid ID/PIN");

playFeedback("error");

return;

}

// ✅ 1.1 Check if this student ID already has an enrolled fingerprint

String indexPath = firebasePath + "/" + idStr + "/finger\_index";

if (Firebase.RTDB.getInt(&firebaseData, indexPath)) {

int existingTemplate = firebaseData.intData();

if (finger.loadModel(existingTemplate) == FINGERPRINT\_OK) {

lcd.clear(); lcd.setCursor(0, 0); lcd.print("Already Enrolled");

lcd.setCursor(0, 1); lcd.print("ID: " + idStr);

playFeedback("warning");

Serial.println("⚠️ Student " + idStr + " already has template " + String(existingTemplate));

return;

}

}

// 👆 2. First scan

lcd.clear(); lcd.print("Place finger...");

while (finger.getImage() != FINGERPRINT\_OK) delay(50);

if (finger.image2Tz(1) != FINGERPRINT\_OK) {

lcd.clear(); lcd.print("Scan failed (1)");

playFeedback("error");

return;

}

// 🤔 3. Check if this finger is already enrolled

if (finger.fingerSearch() == FINGERPRINT\_OK) {

int existingID = finger.fingerID;

String matchedID = getStudentIDFromTemplate(existingID);

lcd.clear();

lcd.setCursor(0, 0); lcd.print("Already Enrolled");

lcd.setCursor(0, 1); lcd.print("ID: " + matchedID);

Serial.println("⚠️ Duplicate finger → ID: " + matchedID);

playFeedback("warning");

return;

}

// ✋ 4. Second scan

lcd.clear(); lcd.print("Remove finger");

delay(1500);

while (finger.getImage() != FINGERPRINT\_NOFINGER) delay(50);

lcd.clear(); lcd.print("Place again...");

while (finger.getImage() != FINGERPRINT\_OK) delay(50);

if (finger.image2Tz(2) != FINGERPRINT\_OK || finger.createModel() != FINGERPRINT\_OK) {

lcd.clear(); lcd.print("Scan failed (2)");

playFeedback("error");

return;

}

// 🤔 5. Get or assign template ID

int templateID = -1;

if (Firebase.RTDB.getInt(&firebaseData, indexPath)) {

templateID = firebaseData.intData();

Serial.println("ℹ️ Existing index for ID " + idStr + ": " + String(templateID));

} else {

templateID = findFirstFreeTemplateSlot();

if (templateID != -1) {

Firebase.RTDB.setInt(&firebaseData, indexPath, templateID);

Serial.println("🆕 Assigned new index " + String(templateID) + " to ID " + idStr);

} else {

lcd.clear(); lcd.print("No slots left!");

playFeedback("error");

Serial.println("❌ No free slot to assign");

return;

}

}

// 🧽 6. Clear sensor slot just in case (safe overwrite)

finger.deleteModel(templateID);

// 🧪 7. Store in sensor and update mapping

if (finger.storeModel(templateID) == FINGERPRINT\_OK) {

templateToID[templateID] = id;

Serial.println("✅ Finger stored → Template: " + String(templateID) + " | ID: " + idStr);

lcd.clear(); lcd.print("Enrolled!");

lcd.setCursor(0, 1); lcd.print("ID: " + idStr);

playFeedback("success");

} else {

lcd.clear(); lcd.print("Store Failed");

playFeedback("error");

Serial.println("❌ Failed to store finger at template ID: " + String(templateID));

}

yield();

}

//=================================================

// FUNCTION: handleattendance(int id)

// PURPOSE : Validate fees and mark attendance with time\_in/time\_out and method

//=================================================

void handleattendance(int id) {

// [1] Connectivity Check

if (WiFi.status() != WL\_CONNECTED) connectToWiFi();

if (!Firebase.ready()) {

Firebase.begin(&config, &auth);

Firebase.reconnectWiFi(true);

yield();

}

// [2] Fetch Fee Data

String name = "ID: " + String(id);

String path = firebasePath + "/" + String(id) + "/payments";

Serial.println("🔥 Fetching from path: " + path);

FirebaseJson paymentsJson;

bool success = false;

for (int i = 0; i < 3; i++) {

if (Firebase.RTDB.getJSON(&firebaseData, path)) {

paymentsJson = firebaseData.jsonObject();

success = true;

break;

}

delay(300); yield();

}

if (!success) {

lcd.clear(); lcd.setCursor(0, 0); lcd.print("No Fee Record");

playFeedback("error");

Serial.println("❌ Could not fetch fees for ID: " + String(id));

return;

}

// [3] Count Unpaid Months

String months[] = {

"fee\_jan", "fee\_feb", "fee\_mar", "fee\_apr", "fee\_may", "fee\_jun",

"fee\_jul", "fee\_aug", "fee\_sep", "fee\_oct", "fee\_nov", "fee\_dec"

};

int unpaid = 0;

for (String m : months) {

FirebaseJsonData fee;

if (paymentsJson.get(fee, m)) {

String val = fee.stringValue;

val.toLowerCase();

if (val.startsWith("unpaid")) unpaid++;

}

}

// [4] Show Status & Feedback

String status = "blocked";

if (unpaid >= 2) {

lcd.clear(); lcd.setCursor(0, 0); lcd.print("BLOCKED - Fees");

lcd.setCursor(0, 1); lcd.print("Pending >1 month");

playFeedback("error");

delay(2500);

turnOffAllLEDs();

return;

} else if (unpaid == 1) {

lcd.clear(); lcd.setCursor(0, 0); lcd.print("1 Month Pending");

lcd.setCursor(0, 1); lcd.print(name);

playFeedback("warning");

delay(2000);

status = "present";

} else {

lcd.clear(); lcd.setCursor(0, 0); lcd.print("WELCOME");

lcd.setCursor(0, 1); lcd.print(name);

playFeedback("success");

delay(2000);

status = "present";

}

turnOffAllLEDs(); // Important

// [5] Mark Attendance (IN or OUT)

String today = getCurrentDate();

String now = getCurrentTime();

String attPath = firebasePath + "/" + String(id) + "/attendance/" + today;

bool marked = false, hasTimeOut = false;

if (Firebase.RTDB.getJSON(&firebaseData, attPath)) {

FirebaseJson& j = firebaseData.jsonObject();

FirebaseJsonData timeIn, timeOut;

if (j.get(timeIn, "time\_in") && timeIn.stringValue != "") marked = true;

if (j.get(timeOut, "time\_out") && timeOut.stringValue != "") hasTimeOut = true;

}

if (marked && hasTimeOut) {

lcd.clear();

lcd.setCursor(0, 0); lcd.print(" Already Marked ");

lcd.setCursor(0, 1); lcd.print(" for today ");

playFeedback("warning");

delay(2000);

return;

}

if (marked && !hasTimeOut) {

Firebase.RTDB.setString(&firebaseData, attPath + "/time\_out", now);

lcd.clear(); lcd.setCursor(0, 0); lcd.print("Goodbye,");

lcd.setCursor(0, 1); lcd.print(name);

playFeedback("success");

Serial.println("📤 time\_out updated for ID: " + String(id));

} else {

FirebaseJson attJson;

attJson.set("time\_in", now);

attJson.set("time\_out", now);

attJson.set("method", fingerprintSuccess ? "fingerprint" : "manual");

if (Firebase.RTDB.setJSON(&firebaseData, attPath, &attJson)) {

Serial.println("✅ Attendance marked for ID: " + String(id));

} else {

Serial.println("❌ Failed to write attendance block");

}

}

// [6] Final Cooldown

fingerprintSuccess = true;

delay(300); yield();

}

// ===========================================================

// FUNCTION: checkPushButton()

// PURPOSE : Enroll or delete fingerprints with admin PIN auth

// ===========================================================

void checkPushButton() {

if (!Firebase.ready()) {

Firebase.begin(&config, &auth);

Firebase.reconnectWiFi(true);

delay(200); yield();

}

static unsigned long lastDebounceTime = 0;

const unsigned long debounceDelay = 50;

int buttonState = digitalRead(PUSH\_BUTTON);

if (buttonState == LOW && !pressed && (millis() - lastDebounceTime > debounceDelay)) {

pressStart = millis();

pressed = true;

lastDebounceTime = millis();

}

if (buttonState == HIGH && pressed && (millis() - lastDebounceTime > debounceDelay)) {

unsigned long pressDuration = millis() - pressStart;

pressed = false;

lastDebounceTime = millis();

// 📦 Collect admin ID & PIN

lcd.clear(); lcd.print("Enter ID:");

String idStr = getPinFromKeypad("ID");

lcd.clear(); lcd.print("Admin PIN:");

String pin = getPinFromKeypad("PIN");

String pinPath = firebasePath + "/" + idStr + "/pin";

bool verified = false;

for (int i = 0; i < 3; i++) {

if (Firebase.RTDB.getString(&firebaseData, pinPath)) {

if (firebaseData.stringData() == pin) {

verified = true;

break;

}

}

delay(300 \* (1 << i)); yield();

}

if (!verified) {

lcd.clear(); lcd.print("Invalid Admin");

playFeedback("error");

return;

}

// 🔁 SHORT PRESS → Enroll Fingerprint

if (pressDuration < 1500) {

enrollNewFingerprint(idStr, pin);

}

// 🗑️ LONG PRESS → Delete Fingerprint

else if (pressDuration >= 2500) {

String fingerPath = firebasePath + "/" + idStr + "/finger\_index";

if (Firebase.RTDB.getInt(&firebaseData, fingerPath)) {

int templateID = firebaseData.intData();

lcd.clear(); lcd.setCursor(0, 0); lcd.print("Del Finger:");

lcd.setCursor(0, 1); lcd.print("ID: " + idStr);

delay(1200);

// Step 1: Confirm via keypad

lcd.clear(); lcd.print("Press # to CONF");

unsigned long confirmStart = millis();

bool confirmed = false;

while (millis() - confirmStart < 10000) {

char key = keypad.getKey();

if (key == '#') {

confirmed = true;

break;

}

delay(10);

}

if (!confirmed) {

lcd.clear(); lcd.print("Cancelled");

delay(1000); return;

}

// Step 2: Confirm via long button hold

lcd.clear(); lcd.print("Hold Btn to CONF");

lcd.setCursor(0, 1); lcd.print("ID: " + idStr);

while (digitalRead(PUSH\_BUTTON) == HIGH) delay(10); // wait for press

unsigned long holdStart = millis();

while (digitalRead(PUSH\_BUTTON) == LOW) {

if (millis() - holdStart >= 2000) break;

delay(10);

}

if (millis() - holdStart < 2000) {

lcd.clear(); lcd.print("Cancelled");

delay(1000); return;

}

// 🔥 Final Delete from Sensor + Firebase

if (finger.deleteModel(templateID) == FINGERPRINT\_OK) {

Firebase.RTDB.deleteNode(&firebaseData, fingerPath);

templateToID.erase(templateID);

lcd.clear(); lcd.print("FP Deleted");

lcd.setCursor(0, 1); lcd.print("ID: " + idStr);

playFeedback("success");

Serial.println("✅ Fingerprint deleted for ID: " + idStr);

} else {

lcd.clear(); lcd.print("Delete Failed");

playFeedback("error");

Serial.println("❌ Sensor delete failed for ID: " + idStr);

}

} else {

lcd.clear(); lcd.print("No FP Found");

playFeedback("warning");

Serial.println("⚠️ No finger\_index for ID: " + idStr);

}

}

}

}

// ===================================================

// FUNCTION: handlePINLogin()

// PURPOSE : Authenticate student via PIN and log attendance

// Triggered when \* or # is pressed on keypad

// ===================================================

void handlePINLogin() {

isTypingPIN = true;

lcd.clear(); lcd.setCursor(0, 0); lcd.print("Enter ID:");

String idStr = getPinFromKeypad("ID");

int id = idStr.toInt();

lcd.clear(); lcd.setCursor(0, 0); lcd.print("Enter PIN:");

String pin = getPinFromKeypad("PIN");

lcd.clear(); lcd.setCursor(0, 0); lcd.print("Checking...");

delay(100); yield();

// Ensure Firebase is ready

if (!Firebase.ready()) {

Firebase.begin(&config, &auth);

Firebase.reconnectWiFi(true);

delay(200); yield();

}

// 🔐 Verify PIN from Firebase with retry

String path = firebasePath + "/" + idStr + "/pin";

String firebasePIN = "";

bool verified = false;

for (int i = 0; i < 3; i++) {

if (Firebase.RTDB.getString(&firebaseData, path)) {

firebasePIN = firebaseData.stringData();

verified = true;

break;

}

delay(300 \* (1 << i)); yield(); // exponential backoff

}

if (!verified || firebasePIN == "") {

lcd.clear(); lcd.print("ID Not Found");

playFeedback("error");

isTypingPIN = false;

return;

}

if (firebasePIN == pin) {

fingerprintSuccess = false; // indicates manual login

handleattendance(id);

} else {

lcd.clear(); lcd.print("Wrong PIN");

playFeedback("error");

delay(1500);

}

isTypingPIN = false;

}

// ==================================================

// FUNCTION: getPinFromKeypad()

// PURPOSE : Prompt 4-digit PIN input via keypad with visual & audio feedback

// RETURNS : 4-digit PIN or "" if timed out

// ==================================================

String getPinFromKeypad(String label) {

String pin = "";

unsigned long startTime = millis();

const unsigned long timeout = 15000;

lcd.clear();

lcd.setCursor(0, 0); lcd.print("Enter " + label);

lcd.setCursor(0, 1); lcd.print("PIN: ");

while ((millis() - startTime) < timeout && pin.length() < 4) {

char key = keypad.getKey();

if (key != NO\_KEY) {

// 🧠 Handle digit input

if (key >= '0' && key <= '9') {

playKeyBeep();

pin += key;

lcd.setCursor(5 + pin.length() - 1, 1);

lcd.print("\*");

}

// 🔙 Backspace support using '#' or '\*'

else if ((key == '#' || key == '\*') && pin.length() > 0) {

playKeyBeep();

pin.remove(pin.length() - 1);

lcd.setCursor(5 + pin.length(), 1);

lcd.print(" ");

lcd.setCursor(5 + pin.length(), 1);

}

// ⏱ Reset timeout after key press

startTime = millis();

}

delay(30);

}

if (pin.length() < 4) {

lcd.clear();

lcd.setCursor(3, 0); lcd.print("PIN Timed Out");

playFeedback("error");

delay(1500);

return "";

}

return pin;

}

// ===================================================

// FUNCTION: getStudentIDFromPIN

// PURPOSE : Match entered PIN to student ID in Firebase

// RETURNS : Matching student ID (String) or "" if not found

// ===================================================

String getStudentIDFromPIN(String pin) {

for (int id = 1001; id <= 1127; id++) {

String path = firebasePath + "/" + String(id) + "/pin";

if (Firebase.RTDB.getString(&firebaseData, path)) {

if (firebaseData.stringData() == pin) {

Serial.println("[✅] PIN matched to ID: " + String(id));

return String(id);

}

} else {

Serial.println("[⚠️] Failed to fetch PIN for ID: " + String(id));

}

delay(20); // Reduce Firebase read flood

}

Serial.println("[❌] No student found with PIN: " + pin);

return "";

}

// ============================================================================

// FUNCTION: getAllFeeKeys

// PURPOSE : Fills array with all 12 Firebase fee keys: "fee\_jan", ..., "fee\_dec"

// ============================================================================

void getAllFeeKeys(String arr[12]) {

arr[0] = "fee\_jan";

arr[1] = "fee\_feb";

arr[2] = "fee\_mar";

arr[3] = "fee\_apr";

arr[4] = "fee\_may";

arr[5] = "fee\_jun";

arr[6] = "fee\_jul";

arr[7] = "fee\_aug";

arr[8] = "fee\_sep";

arr[9] = "fee\_oct";

arr[10] = "fee\_nov";

arr[11] = "fee\_dec";

}

// =============================================================

// FUNCTION: showWelcomeScreen()

// PURPOSE : Display welcome message on LCD during system boot

// =============================================================

void showWelcomeScreen() {

// ---------------- Display greeting on LCD ----------------

lcd.clear();

lcd.setCursor(0, 0);

lcd.print(" WELCOME TO");

lcd.setCursor(0, 1);

lcd.print(" SMART INSTITUTE");

// --------------Optional startup feedback---------------

tone(BUZZER, 1000, 200); // Short startup beep (optional)

delay(3000); // Hold message for 3 seconds

lcd.clear(); // Clear screen for next message

}

// ===============================================

// Validate 4-digit PIN entered by user

// - Searches Firebase /students/{id}/PIN for match

// - Returns valid ID if match found, else -1

// ===============================================

int validatePIN(String pin) {

if (Firebase.RTDB.getJSON(&firebaseData, firebasePath)) {

FirebaseJson &data = firebaseData.jsonObject();

size\_t count = data.iteratorBegin();

String key, value;

int type;

for (size\_t i = 0; i < count; i++) {

data.iteratorGet(i, type, key, value); // Correct usage

// Now 'key' is like "1001", "1002", etc.

FirebaseJsonData pinResult;

data.get(pinResult, key + "/PIN");

if (pinResult.success && pinResult.stringValue == pin) {

data.iteratorEnd(); // Good practice to free memory

return key.toInt(); // Return matched ID

}

}

data.iteratorEnd();

}

return -1; // No match found

}

// ==========================================================

// FUNCTION: showWaitingMessage()

// PURPOSE : Alternates LCD display between:

// 1. Current Time and Date

// 2. Scan Finger / Enter PIN Prompt

// Skips display while PIN is being typed.

// ==========================================================

void showWaitingMessage() {

if (isTypingPIN) return; // Don't interrupt active input

static unsigned long lastSwitchTime = 0;

static bool showClock = true;

const unsigned long displayDuration = showClock ? 3000 : 4000; // Time vs Prompt duration

if (millis() - lastSwitchTime >= displayDuration) {

lcd.clear();

if (showClock) {

// -------- Display Time and Date --------

time\_t now = time(nullptr);

struct tm\* t = localtime(&now);

char timeStr[17];

char dateStr[17];

snprintf(timeStr, sizeof(timeStr), "%02d:%02d:%02d", t->tm\_hour, t->tm\_min, t->tm\_sec);

snprintf(dateStr, sizeof(dateStr), "%02d-%02d-%04d", t->tm\_mday, t->tm\_mon + 1, t->tm\_year + 1900);

lcd.setCursor(0, 0);

lcd.print(timeStr);

lcd.setCursor(0, 1);

lcd.print(dateStr);

}

else {

// -------- Display Prompt --------

lcd.setCursor(0, 0);

lcd.print("Scan Finger or");

lcd.setCursor(0, 1);

lcd.print(" # to Enter PIN ");

}

showClock = !showClock;

lastSwitchTime = millis();

}

}

// ==========================================================

// FUNCTION: connectToWiFi()

// PURPOSE : Connect to WiFi with LCD + Serial feedback

// - Shows animated dots during retry

// - Limits to 20 attempts (10 seconds)

// ==========================================================

void connectToWiFi() {

WiFi.begin(ssid, password);

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("Connecting WiFi");

Serial.print("Connecting to WiFi");

int retries = 0;

lcd.setCursor(0, 1);

lcd.print("Connecting");

while (WiFi.status() != WL\_CONNECTED && retries < 20) {

lcd.print(".");

Serial.print(".");

delay(500);

retries++;

// Limit to 16 characters per line

if (retries % 16 == 0) {

lcd.setCursor(0, 1);

lcd.print("Connecting "); // Reset line

}

}

// -------------------- SUCCESS --------------------

if (WiFi.status() == WL\_CONNECTED) {

Serial.println("\n✅ WiFi Connected!");

Serial.print("IP Address: ");

Serial.println(WiFi.localIP());

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("WiFi Connected!");

lcd.setCursor(0, 1);

lcd.print(WiFi.localIP()); // Display IP address

}

// -------------------- FAILURE --------------------

else {

Serial.println("\n❌ WiFi Failed!");

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("WiFi Failed!");

lcd.setCursor(0, 1);

lcd.print("Check Network!");

delay(3000);

}

}

// ===========================================================

// Get Current Date as String

// Format: YYYY-MM-DD

// ===========================================================

String getCurrentDate() {

time\_t now = time(nullptr); // Get current time

struct tm\* t = localtime(&now); // Convert to local time structure

char buf[11]; // Buffer to hold date string

sprintf(buf, "%04d-%02d-%02d",

t->tm\_year + 1900, // Year (adjust from 1900)

t->tm\_mon + 1, // Month (0-11, so +1)

t->tm\_mday); // Day

return String(buf); // Return formatted date

}

// ===========================================================

// Get Current Time as String

// Format: HH:MM:SS

// ===========================================================

String getCurrentTime() {

time\_t now = time(nullptr);

struct tm\* t = localtime(&now);

char buf[9]; // HH:MM:SS

sprintf(buf, "%02d:%02d:%02d", t->tm\_hour, t->tm\_min, t->tm\_sec);

return String(buf);

}

// ===========================================================

// Turn OFF All Status LEDs (Green, Orange, Red)

// ===========================================================

void turnOffAllLEDs() {

digitalWrite(GREEN\_LED, LOW); // Turn off GREEN (Fee OK indicator)

digitalWrite(ORANGE\_LED, LOW); // Turn off ORANGE (1-month due indicator)

digitalWrite(RED\_LED, LOW); // Turn off RED (Blocked or error indicator)

}

// ==================

// FUNCTION: playKeyBeep()

// PURPOSE : Non-blocking audible click when keypad key is pressed

// ==================

void playKeyBeep() {

tone(BUZZER, 1200); // Start tone

delayMicroseconds(50000); // ≈50ms without blocking too long

noTone(BUZZER); // Stop tone

}

// ==================

// FUNCTION: playFeedback(String type)

// ==================

void playFeedback(String type) {

feedbackType = type;

feedbackStart = millis();

feedbackActive = true;

digitalWrite(GREEN\_LED, LOW);

digitalWrite(ORANGE\_LED, LOW);

digitalWrite(RED\_LED, LOW);

noTone(BUZZER);

if (type == "success") {

digitalWrite(GREEN\_LED, HIGH);

tone(BUZZER, 1000);

} else if (type == "warning") {

digitalWrite(ORANGE\_LED, HIGH);

tone(BUZZER, 700);

} else if (type == "error") {

digitalWrite(RED\_LED, HIGH);

tone(BUZZER, 400);

}

}

void updateFeedback() {

if (!feedbackActive) return;

unsigned long now = millis();

if (feedbackType == "success" && now - feedbackStart >= 400) {

noTone(BUZZER); digitalWrite(GREEN\_LED, LOW); feedbackActive = false;

}

else if (feedbackType == "warning" && now - feedbackStart >= 500) {

noTone(BUZZER); digitalWrite(ORANGE\_LED, LOW); feedbackActive = false;

}

else if (feedbackType == "error" && now - feedbackStart >= 800) {

noTone(BUZZER); digitalWrite(RED\_LED, LOW); feedbackActive = false;

}

}