DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to VTU, Belagavi - 590018)
Accredited by NBA, National Assessment & Accreditation Council (NAAC) with 'A' grade

Mini Project Report on

SMART ATTENDANCE SYSTEM

Submitted in partial fulfillment for the award of degree of

BACHELOR OF ENGINEERING
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

Submitted by

Name	USN
Sai Swaroop K Devarmane	1DS20EE060
Syed Rayyan	1DS20EE082
Utkarsh Vashisth	1DS20EE086
Vinit Hanabar	1DS20EE090

Under the Guidance of
Shruti S A
Assistant Professor
Dept. of E&E
Engg. DSCE,
Bengaluru

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI-590018

2022-2023

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to VTU, Belagavi – 590018, Approved by AICTE & ISO 9001:2015 Certified)

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

Mini Project Course Outcomes and Mapping

Course Outcomes: The Students would be able to

CO1	**Identify a topic related to present day challenges.**
CO2	**Analyze technical aspects of the chosen project with a systematic approach to find a feasible solution for the chosen wo**
CO3	**Use/ implement modern Engineering tools/ technologies to get optimized results**
CO4	**Demonstrate an ability to work in teams.**
CO5	**Develop presentation ,communication and report writing skills**

CO'S\PO'S\PSO	CO-PO Mapping														
	PO1	**PO2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	PSO1	PSO2	PSO3
CO1															
CO2															
CO3															
CO4															
CO5															

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to VTU, Belagavi – 590018, Approved by AICTE & ISO 9001:2008 Certified)
Accredited by NBA, National Assessment & Accreditation Council (NAAC) with 'A' grade

Shavige Malleshwara Hills, Kumaraswamy Layout
Bengaluru-560078
2022-2023

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

CERTIFICATE

Certified that the Mini Project report entitled **"Smart Attendance System"** carried out by **Sai Swaroop K Devarmane (1DS20EE060), Syed Ryyan (1DS20EE082), Utkarsh Vashisth (1DS20EE086) Vinit Hanabar (1DS20EE090)** bonafide students of DAYANANDA SAGAR COLLEGE OF ENGINEERING, an autonomous institution affiliated to VTU, Belagavi in partial fulfillment for the award of Degree of Bachelor of Engineering in Electrical and Electronics Engineering during the year 2022-2023. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The Mini Project report has been approved as it satisfies the academic requirements in respect of work prescribed for the Bachelor of Engineering Degree.

Signature of the Guide
Shruti S A
Assistant Professor
Dept. of E&E Engg.
DSCE, Bengaluru

Signature of the HOD
Dr. P. Usha
Professor & HOD
Dept. of E&E Engg.
DSCE, Bengaluru

Name of the Examiner	Signature with date
1……..……..	……..……..
2……..……..	……..……..

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to VTU, Belagavi – 590018, Approved by AICTE & ISO 9001:2015 Certified)
Accredited by NBA, National Assessment & Accreditation Council (NAAC) with 'A' grade

Shavige Malleshwara Hills, Kumaraswamy Layout
Bengaluru-560078
2022-2023

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

DECLARATION

We, Sai Swaroop K Devarmane (1DS20EE060), Syed Ryyan (1DS20EE082), Utkarsh Vashisth (1DS20EE086) Vinit Hanabar (1DS20EE090) ,respectively, hereby declare that the Mini Project work entitled " Smart Attendance System " has been independently done by us under the guidance of 'Shruti S A', Assistant Professor , EEE department and submitted in partial fulfillment of the requirement for the award of the degree of Bachelor of Engineering in Electrical & Electronics Engineering, at Dayananda Sagar College of Engineering, an autonomous institution affiliated to VTU, Belagavi during the academic year 2022-2023.

We further declare that we have not submitted this report either in part or in full to any other university for the award of any degree.

NAMES	USN
Sai Swaroop K Devarmane	1DS20EE060
Syed Ryyan	1DS20EE082
Utkarsh Vashisth	1DS20EE086
Vinit Hanabar	1DS20EE090

PLACE: Bengaluru

DATE:

List of figure

TABLE OF CONTENTS

ABSTRACT

This report presents a comprehensive overview of an IoT-based biometric attendance system developed using Arduino and ESP32. The system leverages the GT511C3 Fingerprint Sensor for user input, allowing enrolled individuals to be identified through their fingerprints and granted access. The project encompasses two separate codes: one for Arduino and another for ESP32, which work collaboratively to achieve the desired functionality. Additionally, to address potential reliability concerns during adverse conditions, a mobile app has been designed to provide an alternative means for attendance recording. The app incorporates location-based access control, enabling users to authenticate their attendance through fingerprint, PIN, or facial recognition methods via their personal mobile devices. The attendance data, along with the username and real-time location, are securely stored in a database. This report provides a detailed and structured account of the hardware and software implementation, system architecture, integration of components, mobile app development, system evaluation, and future improvements. The IoT-based biometric attendance system demonstrates its potential as a reliable and efficient solution for attendance monitoring in various environments, catering to both hardware and mobile app users.

CHAPTER 1

1.1 Background and Motivation

Attendance management is a crucial aspect in various domains, including educational institutions, corporate organizations, and government agencies. Traditional methods of attendance tracking, such as manual paper-based systems or card-based systems, are prone to errors, time-consuming, and lack accuracy. To overcome these limitations, there is a growing demand for automated and reliable attendance monitoring systems.

The motivation behind this project stems from the need to develop an efficient and convenient attendance management system. By incorporating IoT and biometric technologies, the project aims to enhance accuracy, reduce administrative efforts, and provide real-time monitoring of attendance records. Additionally, the project addresses the limitations of hardware-based biometric systems by integrating a mobile app that allows users to record their attendance using personal mobile devices, ensuring accessibility and flexibility. The IoT-based biometric attendance system utilizes the Arduino microcontroller and ESP32 Wi-Fi module as the core components. The GT511C3 Fingerprint Sensor is employed for capturing and recognizing the enrolled user's fingerprints, granting them access upon authentication. The system consists of two separate codes: one executed on the Arduino board and the other on the ESP32 module.

1.2 Overview

In the Arduino code, user data is stored, including the corresponding usernames and fingerprints. When a user presents their fingerprint, the system matches it with the enrolled data and displays the associated username on an LCD screen. Simultaneously, the system sends real-time data, including the username, to the ThingsBoard cloud platform for further processing and analysis.
The ESP32 code is responsible for establishing communication with the ThingsBoard cloud platform and receiving real-time data from the Arduino. The data is processed and integrated with the Wi-Fi module, enabling seamless transmission between the hardware components and the cloud platform.

1.3 Purpose and Objectives

The purpose of this project is to develop an IoT-based biometric attendance system that offers accurate, secure, and real-time monitoring of attendance records. The project aims to address the limitations of traditional attendance systems, such as time-consuming manual processes and the unreliability of hardware-based biometric systems during adverse conditions.

The objectives of the project include:

1. Implementing a robust hardware system using Arduino and the GT511C3 Fingerprint Sensor for fingerprint recognition and user authentication.
2. Developing an Arduino code that efficiently manages user data, performs fingerprint matching, and displays relevant information on an LCD screen.
3. Integrating the ESP32 Wi-Fi module to establish communication between the hardware system and the ThingsBoard cloud platform.
4. Creating an ESP32 code that securely transmits real-time attendance data to the cloud platform for further processing and analysis.
5. Designing a mobile app that provides an alternative means for attendance recording, incorporating location-based access control and multiple authentication methods (fingerprint, PIN, facial recognition).
6. Storing attendance data, including the username and real-time location, in a database for easy retrieval and management.
7. Identifying potential areas for improvement and suggesting future enhancements to enhance the system's functionality and reliability.

By achieving these objectives, the project aims to revolutionize the conventional attendance management process, offering a comprehensive and efficient solution that caters to both hardware and mobile app users.

CHAPTER 2

2.1 Description of the hardware components

2.1.1 Arduino :

The Arduino board is a microcontroller board that acts as the brain of the biometric attendance system. It is responsible for controlling the overall functionality of the system. Arduino boards come in different models, such as Arduino Uno or Arduino Mega, and offer various digital and analog input/output pins for connecting and controlling other electronic components. The Arduino board is programmed using the Arduino IDE (Integrated Development Environment), which allows developers to write and upload code to the board.

Fig 2.1.1 Arduino board

2.1.2 ESP32 :

The ESP32 is a versatile microcontroller module that provides built-in Wi-Fi and Bluetooth capabilities. It serves as the communication bridge between the biometric attendance system and the ThingsBoard cloud platform. The ESP32 connects to the Arduino board via serial communication and transfers real-time attendance data to the cloud. It is programmed using the Arduino IDE or other development environments compatible with the ESP32.

Fig 2.1.2 ESP32

2.1.3 GT511C3 Fingerprint Sensor:

- The GT511C3 Fingerprint Sensor is a highly accurate and reliable biometric sensor specifically designed for fingerprint recognition. It captures and reads fingerprints using its integrated optical sensor and fingerprint recognition algorithm. The sensor communicates with the Arduino board through serial communication using UART (Universal Asynchronous Receiver-Transmitter) protocol. It can store a certain number of enrolled fingerprints and perform fingerprint matching against the stored data for user authentication.
- GT511C3 fingerprint sensor Module and how it works. This sensor is very different form the Capacitive and Ultrasonic Fingerprint sensor that are commonly used in our smart phones. The GT511C3 is an optical Fingerprint sensor, meaning it relies on images of your fingerprint to recognize its pattern. Yes you read that right, the sensor actually has a camera inside it which takes pictures of your fingerprint and then processes these images using powerful in-built ARM Cortex M3 IC. The below image shows the front and back side of the sensor with pinouts.
- As you can see the sensor has a camera (black spot) surrounded by blue LEDs, these LEDs have to be lit up to take a clear image of the fingerprint. These images are then processed and converted into binary value by using the ARM Microcontroller coupled with EEPROM. The module also has a green color SMD LED to indicate power. Each fingerprint image is of 202x258 pixels with a resolution of 450dpi. The sensor can enroll upto 200 fingerprints and for each finger print template it assigns an ID form 0 to 199. Then during detection it can automatically compare the scanned fingerprint with all 200 templates and if a match is found it gives the ID number of that particular fingerprint using the Smack Finger 3.0 Algorithm on the ARM Microcontroller. The sensor can operate from 3.3V to 6V and communicates through Serial communication at 9600. The communication pins (Rx and Tx) is said to be only 3.3V tolerant, however the datasheet does not specify much about it. The pin-out of a GT511C3 FPS is shown below.
- Apart from serial communication the module can also be directly interfaced to computer though USB connection using the pins shown in previous image. Once connected to computer the module can be controlled using the SDK_DEMO.exe application which can be downloaded from the link. This application allows the user to enroll/verify/delete fingerprints and also to recognize fingerprints. The software can also help you to read the image captured by the sensor which is worth giving it a try. Alternatively you can also use this Software even if the sensor is connected with Arduino,

Fig 2.1.3 FPS - GT511C3

PINOUTS :

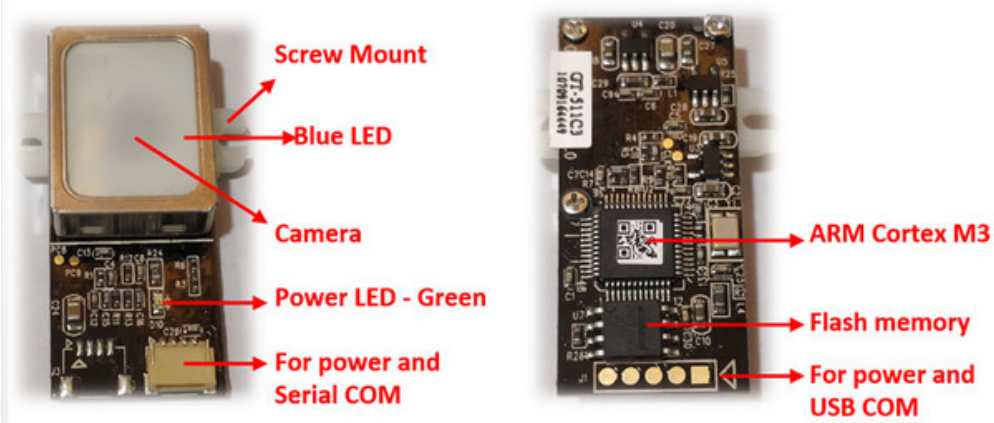


Fig 2.1.4 FPS - Pinouts

2.1.4 Breadboard:

A breadboard is a prototyping board used to connect and test electronic components without the need for soldering. It consists of multiple interconnected metal clips or holes arranged in a grid pattern. Breadboards allow easy and temporary placement of components and provide electrical connectivity through these clips. They are commonly used in the initial stages of hardware prototyping to quickly assemble and test circuits before finalizing the design.

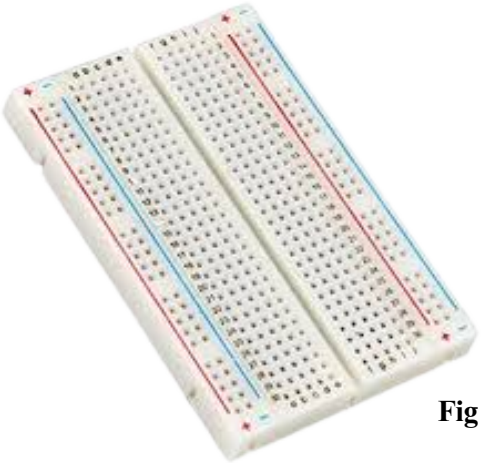


Fig 2.1.3 Breadboard

2.1.5 Resistors (10k and 22k):

Resistors are passive electronic components that resist the flow of electric current. In the biometric attendance system, a 10k resistor is often used as a pull-up resistor for the button. It ensures that the button's input pin is in a known state when the button is not pressed, preventing it from floating and providing a stable input signal. A 22k resistor may be used in voltage divider circuits to scale down voltages or as part of other circuit configurations depending on the specific requirements of the project.

Fig 2.1.4 Resistor

2.1.6 LCD (Liquid Crystal Display) :

An LCD is an output device that visually displays information to the user. It consists of a series of liquid crystals sandwiched between two transparent electrodes. When an electric current is applied, the liquid crystals align to block or allow light, creating visible characters or graphics. In the biometric attendance system, an LCD is used to display the username associated with the recognized fingerprint. It is connected to the Arduino board using digital input/output pins or via an LCD module that simplifies the connection process.

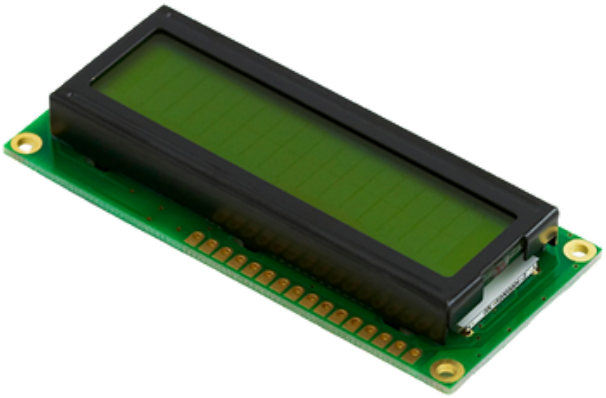


Fig 2.1.6 Liquid Crystal Display

2.1.7 Push Button :

A button is an input device used to trigger a specific action in the system. In the biometric attendance system, a button is utilized to initiate the fingerprint recognition process. When the user places their finger on the GT511C3 Fingerprint Sensor, they simultaneously press the button to signal the system to capture the fingerprint and start the authentication process. The button is connected to a digital input pin on the Arduino board, and its state is monitored to detect when it is pressed.

Fig 2.1.7 Push Button

2.1.8 Jumper Wires :

Jumper wires are flexible wires with pins or connectors at both ends used to establish electrical connections between electronic components. They are vital for interconnecting the Arduino board, GT511C3 Fingerprint Sensor, LCD, button, and other elements on the breadboard. Jumper wires come in different lengths, colors, and types (such as male-to-male, female-to-female, and male-to-female) and facilitate easy and temporary connections, enabling prototyping, testing, and reconfiguration of the hardware system.

Fig 2.1.8 Jumper Wires

2.2 Software Components:

2.2.1 Arduino Code :

1. The Arduino code is responsible for managing the functionality of the biometric attendance system on the Arduino board. It is written in the Arduino programming language, which is a simplified version of C++. The code performs the following tasks:
   - Initialization: It initializes the necessary pins, sets up the communication protocols, and configures the GT511C3 Fingerprint Sensor and LCD.
   - User Data Management: It stores the user data, including usernames and corresponding fingerprints, in variables or data structures. This data can be hardcoded or dynamically updated.
   - Fingerprint Capture: It captures the fingerprint data from the GT511C3 Fingerprint Sensor when the button is pressed and sends it for further processing.
   - Fingerprint Matching: It compares the captured fingerprint with the enrolled fingerprints stored in the user data. It utilizes the fingerprint recognition algorithm provided by the GT511C3 Fingerprint Sensor to perform the matching process.
   - LCD Display: It sends the recognized username to the LCD screen for display, providing immediate feedback to the user about the successful authentication.
   - Data Transmission: It establishes communication with the ESP32 module and sends real-time attendance data, including the username, to the ThingsBoard cloud platform for further processing and storage.

2.2.2 ESP32 Code :

- The ESP32 code is responsible for handling the communication between the biometric attendance system and the ThingsBoard cloud platform. It is typically written in C++ or a programming language compatible with the ESP32 development environment. The code performs the following tasks:
- Initialization: It sets up the necessary Wi-Fi and network configurations on the ESP32 module, establishing a connection with the Wi-Fi network.
- Communication with Arduino: It establishes serial communication with the Arduino board to receive real-time attendance data, including the username.
- Data Processing: It processes the received data, prepares it in the required format, and packages it for transmission to the ThingsBoard cloud platform.
- Cloud Communication: It establishes a secure connection with the ThingsBoard cloud platform using protocols such as MQTT (Message Queuing Telemetry Transport) or HTTP (Hypertext Transfer Protocol).
- Data Transmission: It transmits the attendance data to the ThingsBoard cloud platform, which can further analyze, store, and visualize the data for attendance tracking and reporting.

2.2.3 Mobile App :

- The mobile app is developed for users to record their attendance using their personal mobile devices. It provides an alternative means of attendance recording, integrating location-based access control and multiple authentication methods (fingerprint, PIN, facial recognition). The mobile app typically consists of the following components:
- User Interface: It provides a user-friendly interface for users to interact with the app, including login screens, attendance recording screens, attendance history.
- Location-based Access Control: It incorporates GPS (Global Positioning System) or other location tracking mechanisms to verify the user's presence within a predefined location range (e.g., college premises) before allowing attendance recording.
- Authentication Methods: It integrates biometric authentication methods (fingerprint, facial recognition) or PIN entry for user verification and attendance recording.
- Real-time Data Transmission: It establishes a secure connection with the server or cloud platform to transmit attendance data, including the username and location, in real-time.
- Database Integration: It communicates with the backend database to store attendance records securely and allow for easy retrieval and management.
- Additional Features: In future the mobile app may include additional features such as notifications, statistical analysis, and user management for administrators.
- The software components (Arduino code, ESP32 code, mobile app) work together to provide a seamless and integrated biometric attendance system. The Arduino code handles the hardware interactions, fingerprint recognition, and data display. The ESP32 code facilitates communication with the cloud platform for data transmission and processing. The mobile app offers an alternative means for attendance recording, incorporating location-based access control and multiple authentication methods for user convenience.

2.3 The interaction between hardware and software modules :

The interaction between hardware and software modules in the IoT-based biometric attendance system involves a coordinated flow of data and control signals. Let's examine the interaction at different stages :

1. Initialization:
   - Hardware: The Arduino board initializes the hardware components such as the GT511C3 Fingerprint Sensor, LCD, and button by configuring their respective pins and settings.
   - Software: The Arduino code initializes the software modules, establishing communication with the hardware components and configuring their operation.
2. User Data Management:
   - Hardware: The GT511C3 Fingerprint Sensor captures the fingerprint data of enrolled users and stores it in its internal memory.
   - Software: The Arduino code manages the user data by storing the usernames and corresponding fingerprints in variables or data structures, allowing for later matching during the attendance process.
3. Fingerprint Recognition:
   - Hardware: When the user places their finger on the GT511C3 Fingerprint Sensor and presses the button, the sensor captures the fingerprint data and sends it to the Arduino board.
   - Software: The Arduino code receives the fingerprint data from the sensor and performs the fingerprint matching process. It compares the captured fingerprint with the enrolled user data to determine the user's identity.
4. Display Feedback:
   - Hardware: The Arduino board communicates with the LCD to display the recognized username corresponding to the authenticated fingerprint.
   - Software: The Arduino code sends the username information to the LCD module, which then displays it on the screen, providing immediate feedback to the user about the successful authentication.
5. Data Transmission to Cloud:
   - Hardware: The Arduino board communicates with the ESP32 module through serial communication to transmit attendance data.
   - Software: The Arduino code sends the real-time attendance data, including the username, to the ESP32 module. The ESP32 code receives the data and prepares it for transmission to the ThingsBoard cloud platform using appropriate communication protocols (e.g., MQTT, HTTP).

6. Cloud Processing and Storage:

- Hardware: The ESP32 module establishes a secure connection with the ThingsBoard cloud platform and transmits the attendance data.
- Software: The ESP32 code sends the attendance data to the ThingsBoard cloud platform, where it is received and processed. The cloud platform stores the data securely in a database for further analysis, reporting, and tracking purposes.

The interaction between hardware and software modules is crucial for the seamless operation of the biometric attendance system. The hardware components capture and process data, while the software modules facilitate control, decision-making, and communication between the different components. This synergy enables real-time attendance tracking, data analysis, and cloud-based storage and management.

CHAPTER 3

3.1 Detailed explanation of the FPS code :

The provided Arduino code implements various functionalities of the biometric attendance system. Let's explore each section in detail:

1. Library Inclusion and Initialization:
   - The required libraries, such as Wire, hd44780, FPS_GT511C3, SoftwareSerial, and LiquidCrystal, are included.
   - The LCD display and fingerprint sensor objects are initialized.
2. User Enrollment (Enroll() function):
   - The Enroll() function allows users to enroll their fingerprints into the system.
   - It checks for an available enrollment ID and starts the enrollment process on the fingerprint sensor.
   - The LCD displays instructions for the user, such as "Enroll #" and "Remove finger" at appropriate steps.
   - The fingerprint is captured three times to create a reliable template for enrollment.
   - If the enrollment is successful, the LCD displays "Enrolling Success" and the associated user's name from the Name_List array is displayed.
   - The user's name is stored in the global variable "userName".
3. Main Loop:
   - The main loop starts by checking if the button connected to pin 2 is pressed. If pressed, the Enroll() function is called.
   - If a finger is detected on the fingerprint sensor (fps.IsPressFinger() is true), the captured fingerprint is compared against enrolled fingerprints.
   - If a match is found (identified by an ID less than 5), the associated user's name is displayed on the LCD.
   - The user's name is stored in the global variable "userName".
   - If no finger is detected, the LCD displays a default message.
   - The user's name (stored in "userName") is sent to the ESP32 module via the ESP serial connection for further processing.
4. Communication with ESP32:
   - The Arduino code communicates with the ESP32 module using the SoftwareSerial library (RX on pin 0, TX on pin 1).
   - The user's name (stored in "userName") is sent to the ESP32 using ESP.println(userName).
   - After sending the data, there is a delay of 1 second.
   - If there is data available from the ESP32 (ESP.available()), it is read as a response.
   - The response can be processed as needed. In the provided code, the response is printed to the serial monitor.

The Arduino code manages user enrollment, fingerprint recognition, LCD display, and data transmission to the ESP32 module, demonstrating hardware-software interaction for the biometric attendance system.

3.2 Detailed explanation of the ESP32 code :

The provided ESP32 code connects to a Wi-Fi network, establishes a connection with the ThingsBoard cloud platform, and sends data for real-time monitoring. Let's analyze the different sections:

1. Library inclusion and variable declaration:
   - Libraries like WiFi.h and PubSubClient.h are included.
   - Constants for parameters such as RXp2, TXp2, WIFI_SSID, WIFI_PASSWORD, TOKEN, and THINGSBOARD_SERVER are defined.
   - The wifiClient object is created.
2. Setup():
   - Serial communication is initiated with a baud rate of 9600.
   - Serial2 is set up with the specified baud rate and communication pins.
   - The InitWiFi() function is called to connect the ESP32 to the Wi-Fi network.
   - ThingsBoard server and port are set using client.setServer().
   - The lastSend variable is initialized to 0.
3. Send_to_Thingsboard():
   - Checks if the client is connected to the ThingsBoard server and reconnects if not.
   - Reads data from Serial2, trims whitespace, and stores it in receivedData.
   - Includes received data in a JSON-formatted payload string.
   - Converts the payload to a char array and publishes it to ThingsBoard using client.publish().
   - Displays the published message in the serial monitor.
4. InitWiFi():
   - Connects the ESP32 to the specified Wi-Fi network.
   - Waits until the connection is established.
   - Displays the IP address in the serial monitor.
5. Reconnect():
   - Attempts to reconnect to the ThingsBoard server using client.connect().
   - Displays a success message if the connection is established.
   - Displays an error message and retries after a 5-second delay if the connection fails.
6. Loop():
   - Checks the elapsed time since the last data send against the sendInterval value.
   - Calls Send_to_Thingsboard() to send data to ThingsBoard if the specified time has passed.
   - Uses client.loop() to maintain the MQTT connection.

In summary, the ESP32 code connects to a Wi-Fi network, establishes an MQTT connection with ThingsBoard, and sends data from the Arduino for real-time monitoring. It handles reconnection attempts and ensures a continuous connection with the platform.

CHAPTER 4

4.1 Interfacing LCD and Arduino using I2C Module :

Interfacing an LCD (Liquid Crystal Display) with Arduino using an I2C (Inter-Integrated Circuit) module allows for simplified and efficient communication between the Arduino board and the LCD. The I2C module reduces the number of pins required for communication, making it easier to connect and use the LCD in your Arduino projects. Here's a step-by-step explanation of how to interface an LCD and Arduino using an I2C module:

1. Gather the Required Components:
   - Arduino board (e.g., Arduino Uno)
   - LCD module (16x2 or 20x4)
   - I2C module (commonly based on the PCF8574 or PCF8574A chip)
   - Jumper wires
2. Wiring Connections:
   - Connect the VCC pin of the I2C module to the 5V pin of the Arduino.
   - Connect the GND pin of the I2C module to the GND pin of the Arduino.
   - Connect the SDA pin of the I2C module to the SDA (A4) pin of the Arduino.
   - Connect the SCL pin of the I2C module to the SCL (A5) pin of the Arduino.
3. Install the Required Libraries :
   - Open the Arduino IDE.
   - Go to "Sketch" -> "Include Library" -> "Manage Libraries".
   - Search for "LiquidCrystal_I2C" and install the library developed by Frank de Brabander.
4. Include the Libraries and Define Constants:
   - Open a new Arduino sketch.
   - Include the LiquidCrystal_I2C library using the following line:
   - #include <LiquidCrystal_I2C.h>
   -
   - Define the I2C address of your LCD module (commonly 0x27 or 0x3F) using the following line:
   - #define LCD_ADDRESS 0x27
   -
5. Initialize the LCD Object:
   - Create an instance of the LiquidCrystal_I2C class with the I2C address and LCD dimensions using the following line:
   - LiquidCrystal_I2C lcd(LCD_ADDRESS, 16, 2); // Adjust the dimensions (16, 2) as per your LCD module

6. Setup Function:

- In the setup function, initialize the LCD module using the begin() function:
- void setup() {
- lcd.begin(16, 2); // Adjust the dimensions (16, 2) as per your LCD module
- }

7. Main Loop:

- In the main loop, you can use various functions of the LiquidCrystal_I2C library to control the LCD, such as displaying text, setting the cursor position, and scrolling the text.
- Here's an example of displaying "Hello, World!" on the LCD:
- void loop() {
- lcd.clear(); // Clear the LCD screen
- lcd.setCursor(0, 0); // Set the cursor to the first column, first row
- lcd.print("Hello, World!"); // Print the text on the LCD
- delay(1000); // Delay for 1 second
- }

8. Upload the Code:

- Connect your Arduino board to your computer using a USB cable.
- Select the appropriate board and port in the Arduino IDE.
- Click on the "Upload" button to upload the code to your Arduino board.

After uploading the code, you should see the text "Hello, World!" displayed on your LCD. You can modify the code to display different messages or incorporate other LCD functions as needed.

4.2 Voltage divider circuit :

A Voltage or Potential Divider Circuit is commonly used circuit in electronics where an input voltage has to be converted to another voltage lower than then the original. This is very useful for all analog circuits where variable voltages are required, hence it is important to understand how this circuit works and how to calculate the values of the resistors required to make a voltage divider circuit to output the desired voltage.

The GT511C3 Fingerprint Sensor operates at a voltage level of 3.3V. However, the Arduino board provides a 5V logic level. To ensure compatibility and prevent damage to the sensor, a voltage divider circuit is used to convert the 5V signal from the Arduino's TX (transmit) pin to 3.3V before it reaches the RX (receive) pin of the GT511C3 sensor.

The voltage divider circuit consists of two resistors: a 10k resistor and a 22k resistor. These resistors are connected in series between the Arduino's TX pin and the GT511C3's RX pin. The RX pin of the GT511C3 sensor is connected to the junction point between the two resistors.

The purpose of the voltage divider circuit is to create a voltage drop across the resistors, resulting in a reduced voltage at the RX pin of the sensor. The resistor values are chosen to achieve the desired voltage level conversion. In this case, the 10k resistor and 22k resistor combination is used to convert the 5V signal from the Arduino's TX pin to approximately 3.3V.

By using this voltage divider circuit, the GT511C3 Fingerprint Sensor can safely receive the 3.3V signal from the Arduino board without being exposed to the higher 5V logic level. This ensures proper communication between the Arduino and the fingerprint sensor, allowing them to exchange data effectively and preventing any potential damage that may occur due to voltage incompatibility.

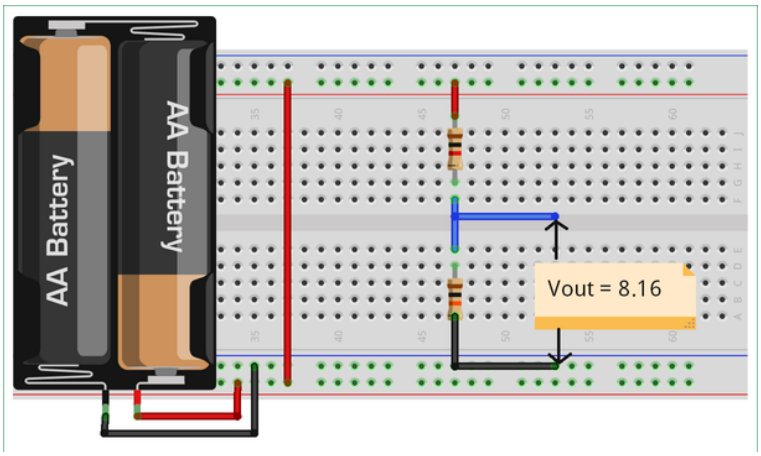


Fig 2.1.4 Voltage Divider Ciruit

4.3 Push Button with Arduino :

To interface a push button with an Arduino board, you would typically follow these steps:

1. Connect the push button to a digital pin on the Arduino:
   - One terminal of the push button should be connected to the digital pin on the Arduino (e.g., D2).
   - The other terminal of the push button should be connected to the ground (GND) pin on the Arduino.
   - Optionally, you can add a pull-up or pull-down resistor for stable button behavior.
2. Configure the digital pin as an input:
   - In your Arduino code, use the pinMode() function to set the pin connected to the push button as an input pin.
   - Example: pinMode(2, INPUT); sets pin D2 as an input pin.
3. Read the state of the push button:
   - Use the digitalRead() function to read the state of the button.
   - Example: int buttonState = digitalRead(2); reads the state of the button connected to pin D2 and stores it in the buttonState variable.
4. Implement the desired functionality based on the button state:
   - Check the value of buttonState to determine if the button is pressed or released.
   - Depending on the button state, you can execute specific code or trigger certain actions.
   - For your described scenario, when the button is pressed (buttonState equals HIGH), you can put the program in enroll mode to allow the user to enroll a new fingerprint.

4.4 ESP32 Interfacing With Arduino :

To interface an ESP32 with an Arduino, you can follow these steps:

1. Connect the ESP32 to the Arduino:
   - Connect the VCC pin of the ESP32 to the 3.3V pin on the Arduino.
   - Connect the GND pin of the ESP32 to the GND pin on the Arduino.
   - Connect the RX pin of the ESP32 to a digital pin on the Arduino (e.g., TX1).
   - Connect the TX pin of the ESP32 to a digital pin on the Arduino (e.g., RX1).
   - Optionally, you may need to use a voltage divider circuit or level shifter to ensure compatibility between the 3.3V logic of the ESP32 and the 5V logic of the Arduino.
2. Install the necessary libraries:
   - In the Arduino IDE, navigate to "Sketch" -> "Include Library" -> "Manage Libraries".
   - Search for and install the necessary libraries for ESP32, such as the WiFi library for wireless connectivity and any additional libraries required for your specific project.

3. Include the libraries and define the necessary objects:
    - In your Arduino code, include the required libraries, such as WiFi.h for WiFi communication.
    - Define an object of the WiFiClient class to establish a connection with the ESP32.
    - You can use additional libraries like the PubSubClient library for MQTT communication with the ESP32.
4. Configure the ESP32 and establish communication:
    - In the setup() function, initialize the serial communication for debugging purposes using Serial.begin().
    - Set up the WiFi connection by calling WiFi.begin() with the appropriate credentials.
    - Optionally, configure additional settings for the WiFi connection, such as static IP or hostname.
    - Establish a connection to the ESP32 by creating an instance of WiFiClient and connecting to the ESP32's IP address or hostname.
5. Implement the main loop:
    - In the loop() function, handle the communication and interaction with the ESP32.
    - Use functions like client.write() or client.println() to send data or commands to the ESP32.
    - Read and process responses from the ESP32 using functions like client.available() and client.read().

It's important to note that the specific implementation may vary based on the communication protocol (e.g., UART, I2C, SPI) and the libraries you choose to use.

4.5 FPS Interfacing With Arduino :

To interface the GT511C3 Fingerprint Sensor (FPS) with an Arduino, you would typically follow these steps:

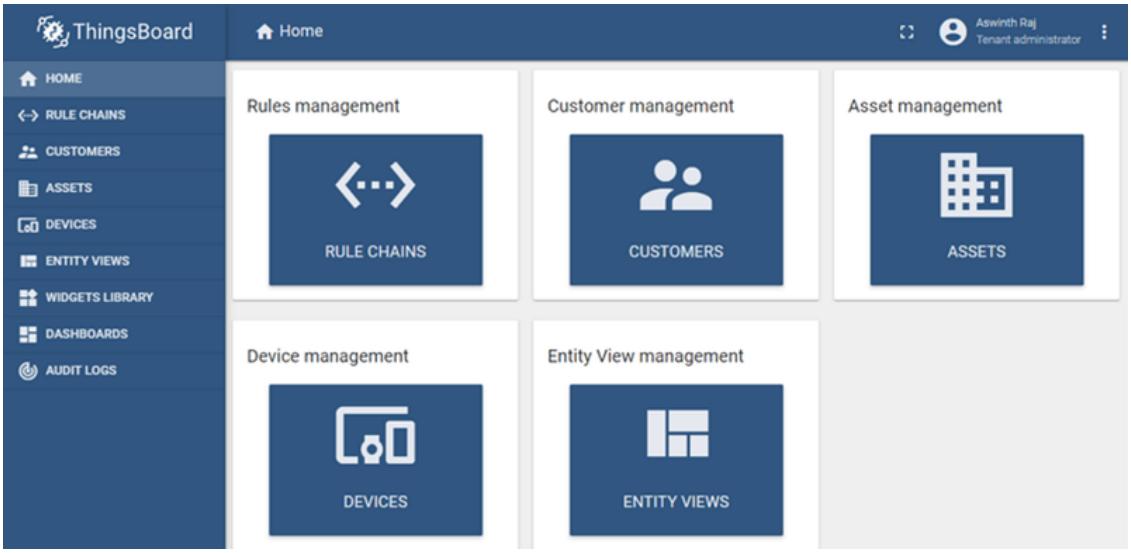
1. Connect the GT511C3 sensor to the Arduino:
    - Connect the VCC pin of the GT511C3 sensor to the 3.3V pin on the Arduino.
    - Connect the GND pin of the GT511C3 sensor to the GND pin on the Arduino.
    - Connect the TX pin of the GT511C3 sensor to a digital pin on the Arduino (e.g., D4).
    - Connect the RX pin of the GT511C3 sensor to a digital pin on the Arduino (e.g., D5).
    - Optionally, you may need to use a voltage divider circuit to ensure the compatibility of signal levels between the Arduino and the GT511C3 sensor (as explained in a previous response).
2. Install the necessary libraries:
    - Download and install the GT511C3 library for Arduino, which provides the necessary functions to communicate with the fingerprint sensor.
    - Install the library by navigating to "Sketch" -> "Include Library" -> "Manage Libraries" in the Arduino IDE and searching for the GT511C3 library.

3. Include the library and define the necessary objects:
    - In your Arduino code, include the GT511C3 library by adding #include <FPS_GT511C3.h> at the beginning of your sketch.
    - Define an object of the FPS_GT511C3 class to interact with the GT511C3 sensor.
    - Example: FPS_GT511C3 fps(4, 5); creates an FPS_GT511C3 object with the RX pin connected to D4 and the TX pin connected to D5.
4. Initialize the sensor and perform required operations:
    - In the setup() function, initialize the serial communication for debugging purposes using Serial.begin().
    - Initialize the GT511C3 sensor by calling fps.Open() to establish communication with the sensor.
    - You can configure additional settings such as LED control using functions provided by the library.
    - Implement desired functionality based on the sensor's capabilities:
        - Enroll new fingerprints using fps.EnrollStart() and related functions.
        - Verify fingerprints using fps.CaptureFinger() and fps.Verify1_N() functions.
        - Retrieve enrolled fingerprint count using fps.GetEnrollCount().
        - Handle responses and perform appropriate actions based on the return values from the sensor.
5. Implement the main loop:
    - In the loop() function, continuously check for fingerprint events or perform other required tasks.
    - Use functions like fps.IsPressFinger() to check if a finger is placed on the sensor.
    - Implement logic to process fingerprint data, such as identifying fingerprints or triggering actions based on recognized fingerprints.

4.6 Preparing your Thingsboard account :

Setting up the Thingsboard account first :

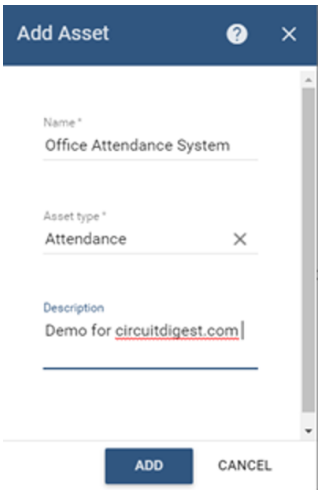
Get into thingboard.io and click on “TRY IT NOW” and then under the community edition (because it’s free) click on Live Demo.



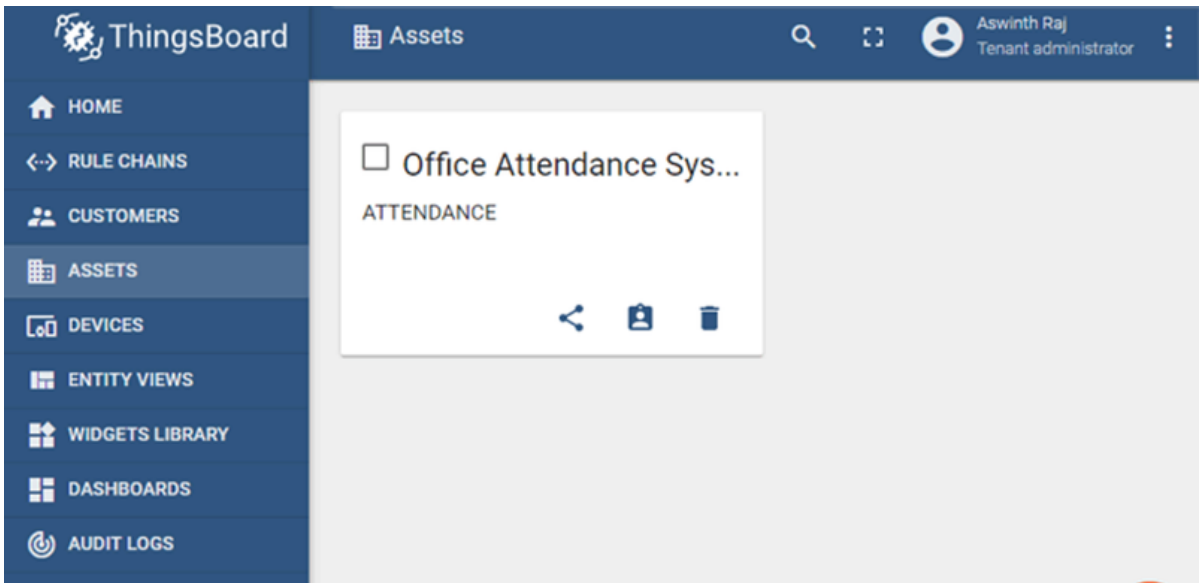
On ThingsBoard we have two important terms, Assets and Devices. You can think of assets as buildings, warehouses, industry, farmland etc and devices as the sensor or devices present in that particular asset. So every asset will have one or more devices in it based on the project, here we will have one asset and one device in our asset.

Creating an Asset on Thingsboard :

Creating our first asset, click on assets on your left panel and you will notice all the assets related to your account, you might have none or some example assets which you can ignore. To create an asset click on the add icon on the bottom right corner of the screen which creates a pop-up promoting for Name, Asset type and Description. You can give any name and type, I have given the following.



Once the details are entered just click on Add and your asset will be created. Note that I have named by asset as , remember this for we will need it later. Once the asset is created you can notice it appearing on the window as shown below.

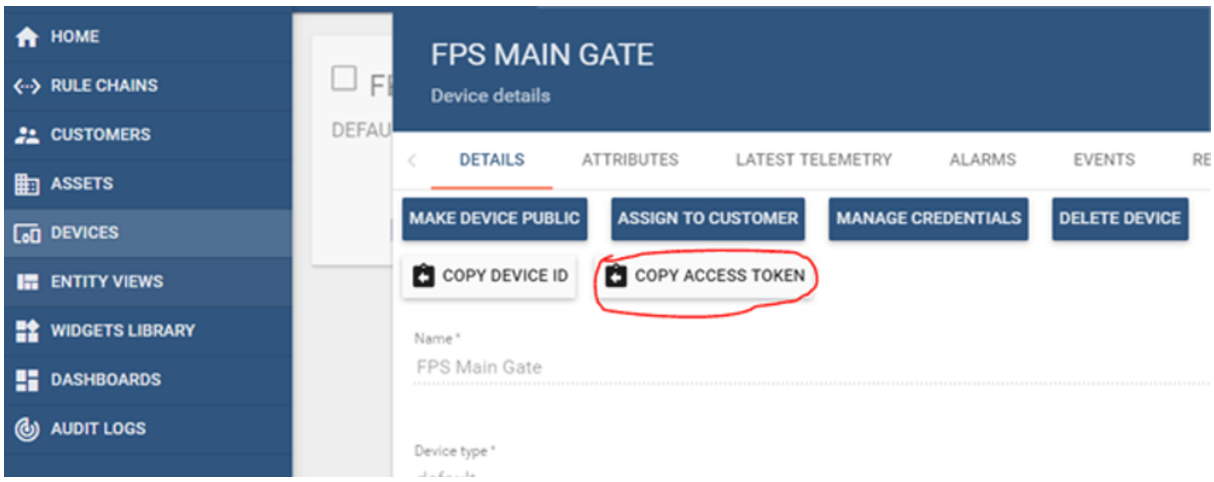


Adding a Device to the Asset :

Now that we have created an asset we should add a device to it. To do so click on the device tab on the left panel and then click on add icon on the bottom right corner of the screen. You will get a similar pop up in which you to name the device I have name mine as FPS main gate and device type as default.

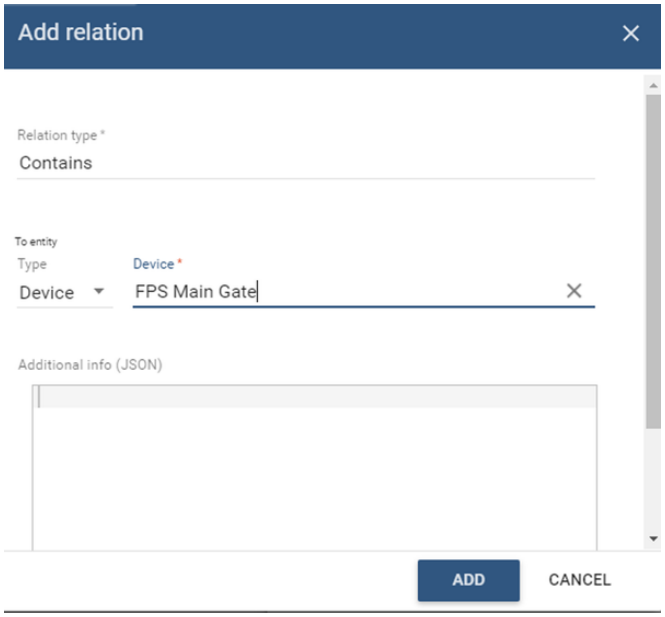
Click on add and then you will find the device being created on the panel, click on the device that we just created and you will notice a new panel sliding in from the right. This panel will have all the information about the device, just click on copy access token as shown below to get the token value of our device, we will need this value in our Arduino program to send or receive values to this device.

Once the details are entered just click on Add and your asset will be created. Note that I have named by asset as , remember this for we will need it later. Once the asset is created you can notice it appearing on the window as shown below.



Creating Device Relation for Asset :

After creating your asset and device, go back to your asset tab and click on the asset that we just created. My asset is named as “Office Attendance System”. This will slide in a pop-up from the right, now select relations tab in the new pop-up and over outbound relations click on add (+) symbol to get the following pop-up



Select entity type as Device and enter the name of the device that we created earlier. My device name was “FPS Main Gate” which I have entered above. Finally click on Add button to add the device relationship to asset.The purpose of this report is to provide an overview of the attendance monitoring app developed for Android. The app aims to streamline the attendance recording process for teachers by leveraging mobile device features such as fingerprint and face recognition. This report will discuss the purpose and features of the app, the location-based access control mechanism, user authentication options, and the real-time attendance recording and data storage capabilities.

Chapter 5

5.1 Overview of the Mobile App's Purpose and Features:

The attendance monitoring app is designed to simplify and automate the process of recording attendance in educational institutions. It provides a convenient and efficient way for teachers to mark attendance using their Android devices. The app offers the following key features:

- Attendance Recording: The app allows teachers to record attendance for their classes directly through their mobile devices.
- User Authentication: The app employs various user authentication options to ensure secure access. Teachers can authenticate themselves using fingerprint recognition, PIN, or facial recognition, depending on the capabilities of their devices.
- Location-based Access Control: The app incorporates a location-based access control mechanism to restrict attendance recording to the college premises. This feature ensures that attendance can only be marked when the teacher's device is within the designated college area.
- Attendance Reports: The app provides teachers with the ability to view their attendance records. Teachers can access detailed reports that display attendance statistics, including the number of classes attended and the percentage of attendance.

5.2 Location-based Access Control Mechanism:

The attendance monitoring app includes a location-based access control mechanism to ensure that attendance can only be recorded when the teacher is within the college premises. This mechanism relies on the device's GPS capabilities to determine the user's location. When the app is launched, it checks the GPS coordinates of the device against a predefined range that represents the college boundaries. If the device is outside this range, attendance recording is disabled.

5.3 User Authentication Options:

To ensure secure access to the app, multiple user authentication options are available:

1. Fingerprint Recognition: Teachers can authenticate themselves using their device's fingerprint scanner. This method provides a quick and convenient way to verify their identity.
2. PIN: Alternatively, teachers can opt to use a PIN code for authentication. This option requires the user to enter a unique PIN to access the app's features.
3. Facial Recognition: If supported by the device, teachers can authenticate themselves using facial recognition technology. This method analyzes the unique facial features of the user to grant access to the app.

5.4 Real-time Attendance Recording and Data Storage:

The attendance monitoring app allows for real-time attendance recording and stores the data securely in a database. When a teacher marks attendance for a class, the app immediately updates the attendance records in the database. This real-time functionality ensures that attendance information is always up to date and readily available for further analysis or reporting.

The app employs a secure database system to store attendance data, ensuring the privacy and integrity of the information. The database can be accessed by authorized personnel to generate attendance reports or perform other administrative tasks related to attendance management.

Chapter 6

6.1 Explanation of how the Arduino, ESP32, and mobile app components interaction

The system integration of the Arduino, ESP32, and mobile app components involves establishing communication and data exchange between these different elements. Here's an overview of how these components interact:

1. Arduino and GT511C3 Fingerprint Sensor:
   - The Arduino board is connected to the GT511C3 Fingerprint Sensor via digital pins.
   - The Arduino code communicates with the fingerprint sensor to perform tasks such as enrolling new fingerprints, capturing and processing fingerprint data, and identifying registered users.
   - When a fingerprint is recognized, the Arduino code retrieves the corresponding user data (such as name) associated with that fingerprint.
2. Arduino and LCD:
   - The Arduino code controls an LCD (Liquid Crystal Display) to provide a visual interface for displaying information.
   - The LCD displays messages, user names, enrollment status, and other relevant information based on the actions performed by the Arduino code and the fingerprint sensor.
3. Arduino and ThingsBoard Cloud Platform:
   - The Arduino code integrates with the ThingsBoard cloud platform to send real-time data.
   - When a user's fingerprint is recognized, the Arduino code sends the corresponding user name and other relevant data to the ThingsBoard cloud platform.
   - The data is transmitted using a network connection (such as Wi-Fi) established by the Arduino board.
4. ESP32 and Mobile App:
   - The ESP32 module is used to enable Wi-Fi connectivity and acts as a bridge between the Arduino and the mobile app.
   - The ESP32 code runs on the ESP32 module and establishes a connection with the Wi-Fi network.
5. Mobile App and User Input:
   - The mobile app provides a user-friendly interface for users to interact with the attendance system.
   - Users can input their attendance information through various means, such as fingerprint recognition, PIN entry, or facial recognition, depending on the options provided by the mobile app.
   - The mobile app may also use location services to verify the user's presence at the designated location.

6. Mobile App and Database:
    - The mobile app stores attendance data, including user names, attendance timestamps, and location information, in a database.
    - The attendance data captured through the mobile app is securely transmitted to the database for storage and future analysis.

Overall, the system integration involves seamless communication and data exchange between the Arduino, ESP32, mobile app, and cloud/database components to enable real-time attendance tracking, data storage, and analysis.

6.2 Data flow between the hardware and software modules :

1. **Hardware Data Flow:**
    - The GT511C3 Fingerprint Sensor captures fingerprint data when a user places their finger on the sensor.
    - The Arduino board reads the fingerprint data from the sensor and performs fingerprint recognition and authentication.
    - The Arduino board retrieves user information associated with the recognized fingerprint, such as the user's name.
    - The LCD display connected to the Arduino board shows the user information, enrollment status, and other relevant messages.

1. **Software Data Flow:**
    - The Arduino code processes the fingerprint data and retrieves the associated user information from its internal storage or memory.
    - The Arduino code sends the user information, such as the user's name, to the ESP32 module.
    - The ESP32 module establishes a Wi-Fi connection and communicates with the ThingsBoard cloud platform.
    - The ESP32 code sends the user information to the ThingsBoard cloud platform for real-time data transmission.
    - The user information, including the user's name, is securely transmitted from the ESP32 to the ThingsBoard cloud platform.
    - The ThingsBoard cloud platform receives and stores the user information in its database for further processing or analysis.

Chapter 7

7.1 Ciruit Diagram :

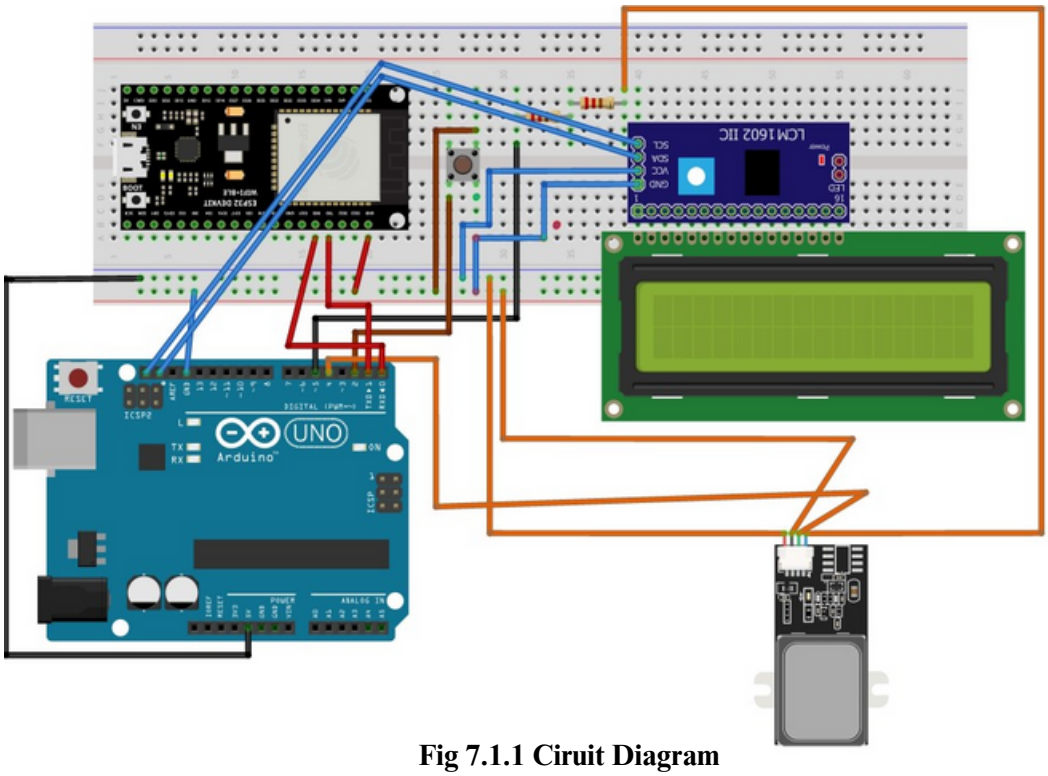


Fig 7.1.1 Ciruit Diagram

7.2 Prototype :

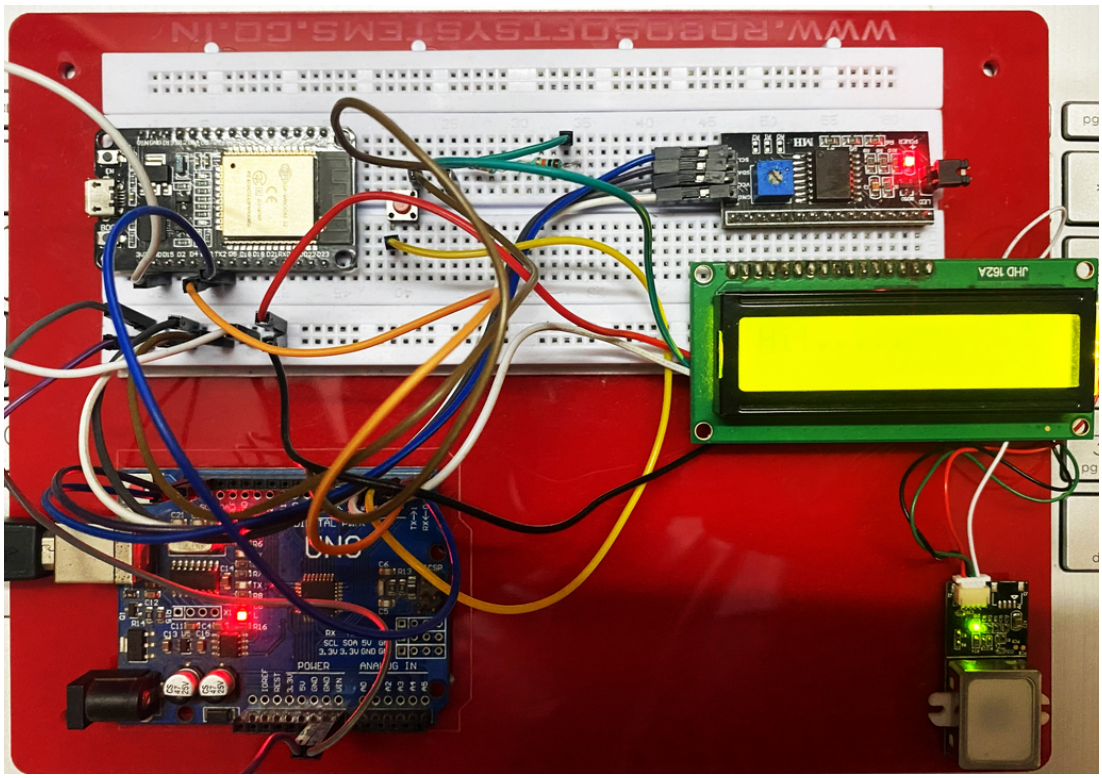


Fig 7.1.2 Prototype

7.3 Block Diagram :

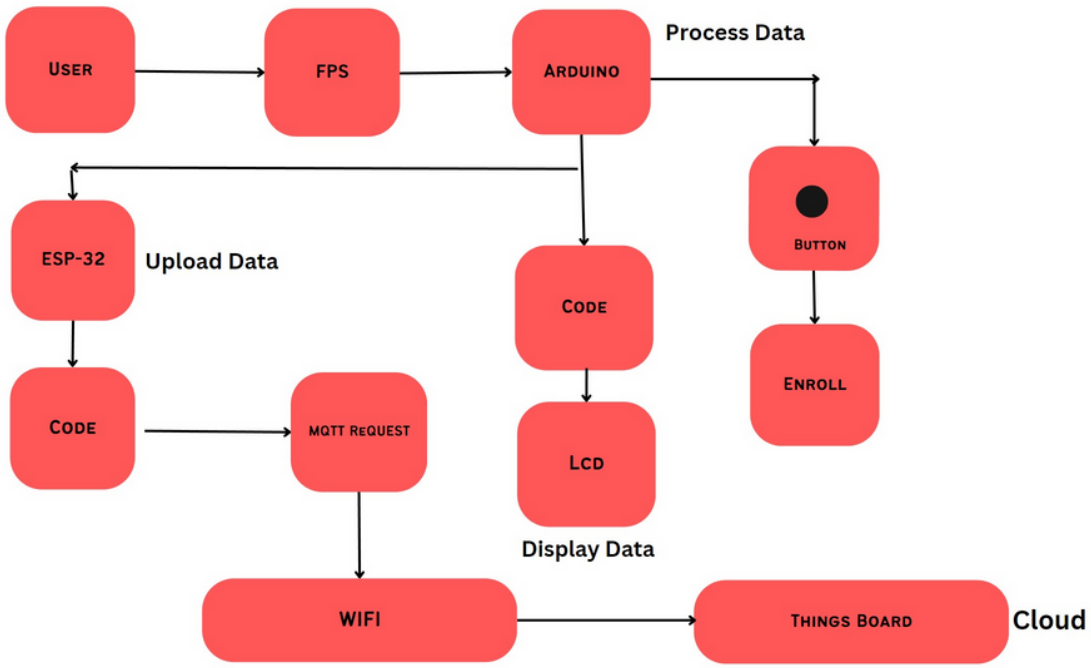


Fig 7.1.3 Block Diagram

7.4 Cloud Data :

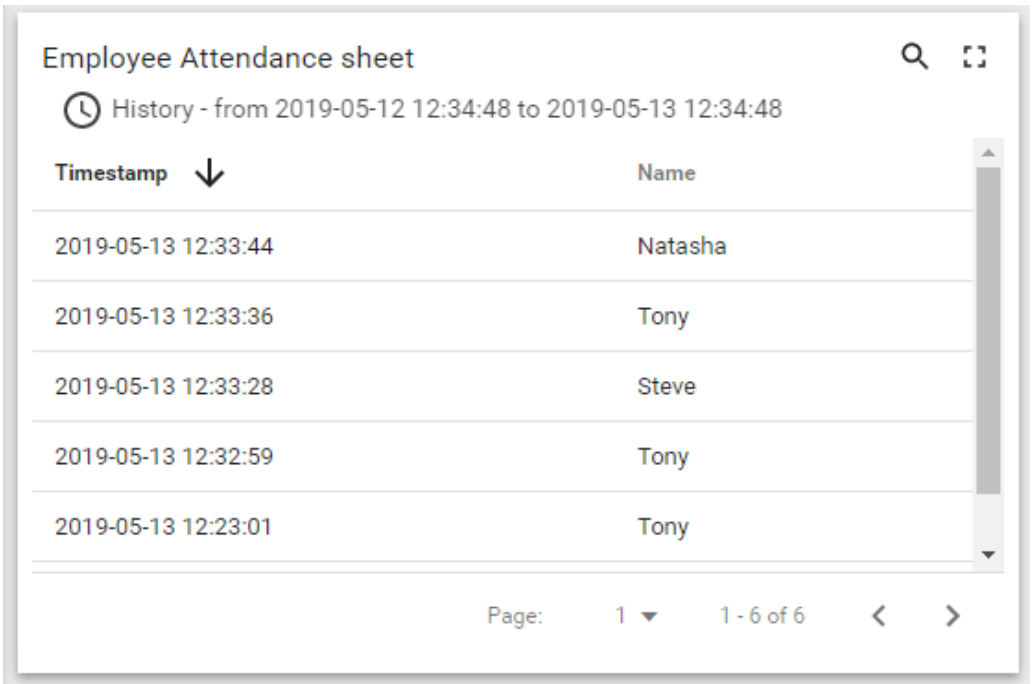


Fig 7.1.4 Data Sheet

7.4 Assessment of the system's performance and reliability

Assessing the performance and reliability of the system is crucial to evaluate its effectiveness in fulfilling its intended purpose. Here are some key points to consider for assessing the system's performance and reliability:

1. Accuracy of Fingerprint Recognition: Evaluate the accuracy of the fingerprint recognition system by testing it with a variety of enrolled fingerprints and verifying the correct identification and authentication of users. Measure the false acceptance rate (FAR) and false rejection rate (FRR) to assess the system's accuracy.
2. Enrollment Process: Assess the ease of enrolling new fingerprints into the system. Test the enrollment process multiple times to ensure its reliability and efficiency. Evaluate whether the system can handle a sufficient number of enrolled users without performance degradation.
3. Real-time Data Transmission: Evaluate the reliability and speed of the data transmission from the Arduino to the ThingsBoard cloud platform via the ESP32 module. Check for any delays or data loss during the transmission process.
4. System Response Time: Measure the system's response time, including fingerprint recognition, user information retrieval, and data transmission. Evaluate whether the response time meets the desired requirements for efficient attendance monitoring.
5. Robustness to Environmental Factors: Assess the system's performance in various environmental conditions, such as temperature variations and humidity levels. Test the system's robustness by simulating challenging scenarios like wet fingers or peeled-off fingerprints.
6. Reliability and Stability: Monitor the system over an extended period to assess its long-term reliability and stability. Look for any issues like software crashes, hardware malfunctions, or inconsistencies in data transmission.
7. User Feedback: Gather feedback from users who have interacted with the system. Assess their satisfaction levels, ease of use, and any suggestions for improvement. User feedback can provide valuable insights into the system's overall performance and reliability.
8. Comparison with Traditional Attendance Systems: Compare the performance and reliability of the IoT-based biometric attendance system with traditional attendance systems, such as manual attendance registers or card-based systems. Evaluate the system's advantages in terms of accuracy, efficiency, and reliability.

By considering these factors and conducting thorough testing and evaluation, you can assess the system's performance and reliability effectively. The results and findings from this assessment will provide valuable insights for further improvements and optimizations if needed.

7.5 Comparison of biometric attendance recording using hardware vs. the mobile app

Biometric attendance recording using hardware and the mobile app offer different approaches to recording and managing attendance. Here is a comparison of both methods:

1. **Hardware-based Biometric Attendance Recording:**
   - Reliability: Hardware-based systems, such as the IoT-based biometric attendance system using Arduino and the GT511C3 Fingerprint Sensor, provide reliable attendance recording. They use biometric characteristics like fingerprints for identification, which are unique to each individual, ensuring accurate attendance tracking.
   - Physical Interaction: Hardware systems require users to physically interact with the biometric sensor, such as placing their finger on the fingerprint sensor. This provides a tangible and secure method of identification.
   - Environmental Limitations: Hardware systems may have limitations based on environmental factors. For example, variations in temperature, humidity, or the condition of the fingers (e.g., wet fingers) can affect the performance of the sensor and result in false rejections or false acceptances.
   - Dedicated Device: Hardware systems typically require dedicated devices (such as the Arduino-based system) for attendance recording, which may involve installation and maintenance efforts.
2. **Mobile App-based Biometric Attendance Recording:**
   - Convenience: Mobile app-based attendance recording offers convenience to users as they can record their attendance using their personal mobile devices. They can access the attendance recording feature anytime and anywhere, eliminating the need for dedicated hardware devices.
   - Multiple Biometric Modalities: Mobile apps can leverage various biometric modalities for attendance recording, including fingerprint, facial recognition, or iris scanning. This flexibility allows users to choose the most convenient method based on their device capabilities.
   - Location-based Access: Mobile apps can incorporate location-based access for attendance recording. Users can be granted access to record their attendance only when their device is within a specific location (e.g., college campus). This adds an additional layer of security and prevents attendance fraud.
   - User Experience: Mobile apps offer a user-friendly interface with intuitive features and customization options. Users can view their attendance records, receive notifications, and access other relevant information through the app.

Both hardware-based and mobile app-based biometric attendance recording have their advantages and considerations. The choice depends on factors such as the specific requirements of the organization, the level of security needed, user convenience, and the available resources for implementation and maintenance. Organizations may opt for hardware-based systems when they require dedicated and reliable attendance recording, while mobile app-based systems offer flexibility and convenience for users on their personal devices.

Chapter 8

8.1 Challenges encountered and future improvements

During the development and implementation of the IoT-based biometric attendance system using Arduino, ESP32, and the GT511C3 Fingerprint Sensor, several challenges may have been encountered. Some of the common challenges and potential future improvements include:

1. Accuracy and Reliability: One of the main challenges is ensuring the accuracy and reliability of the biometric identification process. In some cases, the fingerprint sensor may encounter issues such as false rejections or false acceptances. Future improvements could focus on enhancing the fingerprint recognition algorithm or exploring alternative biometric modalities for more robust identification.
2. Environmental Factors: Environmental factors such as temperature, humidity, or the condition of the fingers (e.g., wet fingers) can affect the performance of the fingerprint sensor. Future improvements may involve implementing techniques to mitigate the impact of these factors, such as using additional sensors or incorporating algorithms to compensate for variations in finger conditions.
3. Power Consumption: Efficient power management is crucial for IoT devices. In this system, both Arduino and ESP32 consume power. Future improvements may involve optimizing power consumption by implementing sleep modes or using low-power components to extend the battery life or reduce power requirements.
4. Scalability and Flexibility: The current system may have limitations in terms of scalability and flexibility. Future improvements can focus on developing a modular and scalable architecture that allows for easy integration of additional sensors, support for multiple devices, and seamless expansion to accommodate a larger number of users or multiple locations.
5. Security and Privacy: Biometric data is sensitive, and ensuring the security and privacy of user information is essential. Future improvements may involve implementing robust encryption techniques, secure communication protocols, and compliance with data protection regulations to protect user data from unauthorized access or breaches.
6. User Interface and Experience: Improving the user interface and experience of the system can enhance its usability and adoption. Future improvements may involve designing intuitive and user-friendly interfaces for both the hardware device and the mobile app, providing clear instructions and feedback during enrollment and attendance recording processes, and incorporating features such as notifications or attendance history for users to easily track their records.
7. Integration with Backend Systems: Integration with backend systems, such as the ThingsBoard cloud platform, may require further improvements for seamless data transmission, storage, and analysis. Future enhancements may involve optimizing data processing and communication protocols to ensure real-time and reliable data transfer between the hardware and software components.

8.2 Final thoughts on the system's potential and practicality

Overall, addressing these challenges and implementing future improvements can enhance the performance, accuracy, scalability, and user experience of the biometric attendance system, making it more robust, efficient, and user-friendly.

The IoT-based biometric attendance system using Arduino, ESP32, and the GT511C3 Fingerprint Sensor has the potential to offer an efficient and reliable solution for attendance monitoring in various settings. By leveraging biometric technology, it provides a more secure and convenient way of recording attendance compared to traditional methods like manual entry or swipe cards.

The system's potential lies in its ability to accurately identify individuals through their unique fingerprints, eliminating the possibility of proxy attendance and ensuring data integrity. The real-time data transmission to the ThingsBoard cloud platform enables seamless monitoring and tracking of attendance records, making it suitable for applications in educational institutions, workplaces, and other organizations where attendance management is critical.

The practicality of the system is evident in its hardware and software integration, allowing for easy deployment and usage. The Arduino and ESP32 components provide the necessary computational power and connectivity, while the GT511C3 Fingerprint Sensor ensures accurate biometric identification. The mobile app complements the hardware device by providing an alternative method for attendance recording, catering to scenarios where the hardware may not be accessible or suitable.

Moreover, the system offers flexibility and scalability, allowing for customization and expansion based on specific requirements. Additional features can be integrated, such as facial recognition or PIN authentication, to accommodate different user preferences and enhance security.

While the system has shown promise, it is important to consider certain factors for its successful implementation. Adequate user training and support are necessary to ensure proper usage and address any challenges that may arise. Additionally, regular maintenance and updates are crucial to keep the system functioning optimally and address any potential issues or vulnerabilities.

In conclusion, the IoT-based biometric attendance system has the potential to revolutionize attendance monitoring by providing a more secure, accurate, and convenient solution. With continuous improvements and adaptations to meet specific needs, the system can be a practical and effective tool for organizations seeking efficient attendance management.

References

[1] Ayush Mahajan , Gaurav Sahu , Aditya Anand, "Attendance system based on Fingerprint Using ARDUINO", B K Birla Institute of Engineering & Technology, Pilani, Rajasthan, India.

[2] Ritum Dutta, Nitesh Kumar, "Smart and Secure Fingerprint Attendance System using Arduino Uno with GSM Alert", Conference: 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS) At: Thoothukudi, India

[3] Wang and Jingli , "The Design of Teaching Management System in Universities Based on Biometrics Identification and the Internet of Things Technology", IEEE 10th International Conference on Computer Science & Education (ICCSE), Cambridge University, UK July 22-24, 2015, pp. 979-982.

[4]Circuitdigest.com/microcontroller-projects/iot-based-biometric-attendance-system-using-arduino-and-thingsboard

[5] O. Shoewuand O.A. Idowu, "Development of Attendance Management System using Biometrics", Pacific Journal of Science and Technology, PP. 300-307, 2012.

[6] R.P.Nimisha, Patel & Mona Gajjar, "Online Students' Attendance Monitoring System in Classroom Using Radio Frequency Identification Technology: A Proposed System Framework", International Journal of Emerging Technology and Advanced Engineering, Volume 2,Issue 2, February, 2012.

[7] Wireless Fingerprint Based Student Attendance System, National Institute of Technology Rourkela, 2010.

Appendix A

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
#include "FPS_GT511C3.h"
#include "SoftwareSerial.h"
#include <LiquidCrystal.h>

hd44780_I2Cexp lcd;
hd44780_I2Cexp lcd2(0x27);

FPS_GT511C3 fps(4, 5);
SoftwareSerial ESP(0,1); // RX, TX

char *Name_List[] = {"Sai", "Shruti", "Vinit", "Utkarsh", "Syed"};
String userName = ""; // Declare a global variable to store the user name

void setup()
{
 Serial.begin(9600);
 ESP.begin(9600);
 lcd.begin(16, 2);
 lcd.print("GT511C3 FPS");
 lcd.setCursor(0, 1);
 lcd.print("with Arduino");
 delay(2000);
 lcd.clear();
 fps.Open();
 fps.SetLED(true);
 pinMode(2, INPUT_PULLUP);
}

void Enroll()
{
 int enrollid = 0;
 bool usedid = true;
while (usedid == true)
 {
 usedid = fps.CheckEnrolled(enrollid);
 if (usedid == true)
 enrollid++; }
fps.EnrollStart(enrollid);
 lcd.clear();
 lcd.print("Enroll #");
 lcd.print(enrollid);
 while (fps.IsPressFinger() == false)
  delay(100);
 bool bret = fps.CaptureFinger(true);
 int iret = 0;
 if (bret != false)
 {
  lcd.clear();
  lcd.print("Remove finger");
  fps.Enroll1();
  while (fps.IsPressFinger() == true)
   delay(100);
  lcd.clear();
  lcd.print("Press again");
  while (fps.IsPressFinger() == false)
   delay(100);
  bret = fps.CaptureFinger(true);
  if (bret != false)
  {
   lcd.clear();
   lcd.print("Remove finger");
   fps.Enroll2();
   while (fps.IsPressFinger() == true)
    delay(100);
   lcd.clear();
   lcd.print("Press yet again");
   while (fps.IsPressFinger() == false)
    delay(100);
   bret = fps.CaptureFinger(true);
   if (bret != false)
   {
    lcd.clear();
    lcd.print("Remove finger");
    iret = fps.Enroll3();
```

```
 if (iret == 0)
       {
        lcd.clear();
        lcd.print("Enrolling Success");
        if (enrollid < 5)
        {
         lcd.setCursor(0, 1);
         lcd.print("Name: ");
         lcd.print(Name_List[enrollid]);
         userName = Name_List[enrollid]; // Store
the user name in the global variable
         Serial.print("User Name: ");
         Serial.println(userName);
         // Send userName to ESP32
         ESP.println(userName);
        }
       }
       else
       {
        lcd.clear();
        lcd.print("Enroll Failed:");
        lcd.print(iret);
       }
     }
     else
      lcd.print("Failed 1");
   }
   else
    lcd.print("Failed 2");
 }
 else
  lcd.print("Failed 3");
}

void loop()
{

delay(1500);
 if (digitalRead(2) == 0)
{
  Enroll();
 }
 if (fps.IsPressFinger())
 {
  fps.CaptureFinger(false);
  int id = fps.Identify1_N();
  lcd.clear();
  Serial.println(Name_List[id]);
  lcd.print("Welcome: ");
  if (id == 200)
  {
   lcd.print("Unknown");
  }
  else if (id < 5)
  {

   lcd.print(Name_List[id]);
      userName = Name_List[id]; // Store the
user name in the global variable
    Serial.print("User Name: ");
    Serial.println(userName);
  }
  else
  {
   lcd.print("Invalid ID");
   delay(1000);
  }
 }
 else
 {
  lcd.clear();
  lcd.print("Hi!.....");
 }
 delay(1000);
}
```

Appendix B

```
#include <WiFi.h>
#include <PubSubClient.h>

#define RXp2 16
#define TXp2 17

const char* WIFI_SSID = "Sai";
const char* WIFI_PASSWORD = "saisaisai";
const char* TOKEN = "H2xUiGv2qLp4HVEtEd62";
const char* THINGSBOARD_SERVER = "demo.thingsboard.io";

WiFiClient wifiClient;
PubSubClient client(wifiClient);

unsigned long lastSend = 0;
String userName = ""; // Declare a global variable to store the user name

void setup() {
 Serial.begin(9600);

 delay(10);
    Serial2.begin(9600, SERIAL_8N1, RXp2, TXp2);
 InitWiFi();
   client.setServer(THINGSBOARD_SERVER, 1883);
 lastSend = 0;
}

void Send_to_Thingsboard() {
 if (!client.connected()) {
   reconnect();
 }
if (Serial2.available()) {
   String receivedData = Serial2.readString();
   receivedData.trim();
 Serial.print("Received Data: ");
   Serial.println(receivedData);

      Serial.println("Collecting temperature data.");

   String payload = "{";
          payload += "\"Name\":\"" + receivedData + "\""; // Include the received data in the payload
   payload += "}";

   char attributes[100];
   payload.toCharArray(attributes, 100);
    client.publish("v1/devices/me/telemetry", attributes);
         Serial.println("Data published to ThingsBoard.");
 }
}

void InitWiFi() {
 Serial.println("Connecting to WiFi...");
                  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
            while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
 }
 Serial.println();
 Serial.println("WiFi connected");
 Serial.print("IP address: ");
 Serial.println(WiFi.localIP());
}
void reconnect() {
 while (!client.connected()) {
          Serial.print("Attempting MQTT connection...");
```

```
void reconnect() {
 while (!client.connected()) {
            Serial.print("Attempting            MQTT
connection...");
if   (client.connect("ESP32   Device",   TOKEN,
NULL)) {
     Serial.println("Connected to ThingsBoard");
   } else {
     Serial.print("Failed to connect [");
     Serial.print(client.state());
     Serial.println("] Retrying in 5 seconds...");
     delay(5000);
   }
  }
}

void loop() {
  unsigned long currentMillis = millis();
   unsigned long sendInterval = 5000; // Adjust
this value as needed

  if (currentMillis - lastSend >= sendInterval) {
    Send_to_Thingsboard();
    lastSend = currentMillis;
  }

  client.loop();
}
```