CHAPTER 1

INTRODUCTION

The advancement of Internet of Things (IoT) technology has significantly impacted various sectors, including education. One notable innovation is the use of smart lockers for managing exam papers. These IoT-enabled smart lockers offer a secure, efficient, and streamlined solution for storing and distributing exam materials. They incorporate advanced features such as biometric authentication, sensors for real-time monitoring, and connectivity for centralized management. With these technologies, access to exam papers can be strictly controlled and logged, ensuring only authorized personnel can handle them. Additionally, environmental sensors can maintain optimal storage conditions, while real-time alerts enhance security by notifying administrators of any unauthorized access or anomalies. By automating many aspects of storage and access, smart lockers reduce administrative burdens and improve efficiency. The integration of IoT in smart lockers thus provides a robust solution for maintaining the integrity and confidentiality of exam papers, reflecting the potential of IoT to enhance educational infrastructure.

Figure 1.1: Smart locker for exam paper.

The connectivity aspect of IoT allows for centralized management and remote monitoring of all lockers, making it easier for administrators to oversee the distribution and collection of exam papers. This connectivity also supports data analytics, enabling institutions to optimize locker usage and enhance security protocols based on usage patterns.

Implementing smart lockers not only enhances security but also improves operational efficiency by automating access management and reducing the need for manual oversight. This automation allows educational staff to focus on more critical tasks, ultimately contributing to a more organized and reliable examination process.

In conclusion, IoT-enabled smart lockers represent a significant leap forward in the management of exam papers. By leveraging advanced technologies, these systems provide a robust solution that enhances security, efficiency, and convenience, ensuring

that educational institutions can uphold the highest standards of integrity and confidentiality in their examination processes.

1.1: Problem Statement:

The traditional methods of managing exam papers in educational institutions face several critical challenges, including ensuring the security and integrity of the documents, maintaining confidentiality, and efficiently organizing the distribution and collection process. These issues are exacerbated by the potential for unauthorized access, environmental conditions that may affect the documents, and the administrative burden on staff. To address these problems, there is a need for a modern, technology-driven solution that leverages advancements in Internet of Things (IoT) technology to provide a secure, efficient, and streamlined system for storing and managing exam papers.

Key Components and Their Roles

1. NodeMCU (ESP8266/ESP32)
    - Central Controller: Acts as the brain of the system, managing all other components and handling communication with the central server or cloud platform.
2. Relay
    - Lock Control: Controls the DC motor that locks and unlocks the locker. The NodeMCU triggers the relay to either connect or disconnect the power to the motor.
3. OLED Display
    - User Interface: Provides a visual interface for the user to see status messages, instructions, and confirmations. It can display prompts for fingerprint scanning, successful authentications, and error messages.
4. Fingerprint Sensor
    - Authentication: Scans the user's fingerprint to verify their identity. The sensor sends the fingerprint data to the NodeMCU for authentication.
5. DC Motor

- Locker Mechanism: Operates the physical lock of the locker. When activated by the relay, it either locks or unlocks the locker door.

6. Battery
    - Power Supply: Provides power to the entire system, ensuring it operates even during power outages.

System Overview

1. Initialization
    - The NodeMCU initializes all components: sets up Wi-Fi connectivity, configures the relay, initializes the OLED display, and sets up the fingerprint sensor.
2. User Interaction
    - The OLED display provides instructions to the user, such as prompting them to place their finger on the fingerprint sensor.
3. Authentication Process
    - The user places their finger on the fingerprint sensor.
    - The fingerprint sensor captures the fingerprint and sends the data to the NodeMCU.
    - The NodeMCU processes the fingerprint data and compares it with stored fingerprints to verify the user's identity.
4. Locker Access
    - If authentication is successful, the NodeMCU activates the relay.
    - The relay powers the DC motor, which unlocks the locker.
    - The OLED display confirms that the locker is unlocked.
5. Monitoring and Logging
    - The NodeMCU logs the access event and sends the data to a central server or cloud platform for monitoring and record-keeping.

- If an unauthorized access attempt is detected, the system sends an alert to the administrator.

6. Power Management
    - The battery ensures the system remains operational even if the main power supply fails.
    - The NodeMCU can monitor battery levels and alert the administrator if the battery needs recharging or replacement.

Implementation Details

- Wiring:
    - Connect the NodeMCU to the relay module, fingerprint sensor, OLED display, and DC motor according to their pin configurations.
    - Ensure the battery is properly connected to provide power to the NodeMCU and other components.
- Programming:
    - Use the Arduino IDE to program the NodeMCU. Libraries for handling the OLED display, fingerprint sensor, and Wi-Fi connectivity will be required.
    - Implement code for initializing components, handling user input, processing fingerprint data, controlling the relay, and managing Wi-Fi communication.
- Security:
    - Use encryption for data transmission between the NodeMCU and the central server.
    - Store fingerprint data securely on the NodeMCU, ensuring it cannot be easily accessed or tampered.

1.2: Problem Scope:

The traditional methods of managing exam papers in educational institutions are fraught with several significant challenges, ranging from ensuring document security to efficient handling and distribution. The following outlines the detailed scope of the problems addressed by implementing an IoT-enabled smart locker system:

1. Security of Exam Papers

- Unauthorized Access: Traditional storage methods (like locked cabinets or manual safes) are prone to unauthorized access due to lost or duplicated keys, and insufficient monitoring.
- Tampering and Theft: Physical security measures can be breached, leading to potential tampering or theft of exam papers, compromising the integrity of the examination process.

Solution:

- Advanced Authentication: Implementing a fingerprint sensor ensures that only authorized personnel can access the exam papers. The use of biometric data significantly reduces the risk of unauthorized access.
- Real-Time Monitoring: The system can log access attempts and send real-time notifications to administrators in case of unauthorized access attempts, enhancing security vigilance.

2. Operational Efficiency

- Manual Handling: Storing and retrieving exam papers manually is time-consuming and prone to human error, which can lead to misplacement or loss of papers.
- Administrative Burden: The traditional process requires significant administrative effort to manage, track, and secure exam papers, diverting resources from other critical tasks.

Solution:

- Automated Access Control: The smart locker system automates the access process, reducing the time and effort required to manage exam papers. This automation also minimizes human error.
- Centralized Management: Administrators can manage and monitor locker status and access logs from a centralized platform, streamlining operations and reducing administrative burden.

3. Environmental Conditions

- Document Degradation: Exam papers are susceptible to damage from environmental factors such as humidity, temperature fluctuations, and pests if not stored in optimal conditions.
- Lack of Monitoring: Traditional storage methods often lack the ability to monitor and control environmental conditions.

Solution:

- Environmental Sensors: Integrating sensors to monitor humidity, temperature, and other environmental conditions ensures that exam papers are stored in optimal conditions, preventing degradation.
- Automated Alerts: The system can send alerts if environmental conditions fall outside of predefined thresholds, allowing for timely intervention to protect the documents.

4. Accountability and Transparent

- Lack of Traceability: Traditional methods often lack detailed access logs, making it difficult to trace who accessed the exam papers and when.
- Discrepancies and Audits: The absence of comprehensive records can lead to discrepancies and complicate audit processes.

Solution:

- Access Logs: The smart locker system keeps detailed logs of all access events, including timestamps and user identities, providing a clear and traceable record.
- Enhanced Transparency: These logs enhance accountability and facilitate audits, ensuring that any issues can be quickly identified and addressed.

5. Reliability and Continuity

- Power Outages: Traditional security systems can be compromised during power outages, leading to potential breaches or loss of access control.
- System Failures: Mechanical failures in traditional locking systems can result in delays and increased security risks.

Solution:

- Battery Backup: The inclusion of a battery ensures that the smart locker system remains operational during power outages, maintaining security and access control.
- Regular Maintenance Alerts: The system can be programmed to send alerts for scheduled maintenance, ensuring continuous and reliable operation.

1.3: Advantages Of smart locker for exam paper.

Implementing IoT-enabled smart lockers for managing exam papers offers numerous advantages, enhancing security, efficiency, and reliability in educational institutions. Here are the key benefits:

1. Enhanced Security

- Advanced Authentication: Biometric authentic ensures that only authorized personnel can access the exam papers, significantly reducing the risk of unauthorized access.

- Real-Time Monitoring: Continuous monitoring and logging of access attempts allow for immediate detection and response to any unauthorized access or suspicious activities.
- Tamper Alerts: The system can send instant alerts if tampering is detected, ensuring prompt action to secure the exam papers.

2. Improved Efficiency

- Automated Access Control: Automation reduces the time and effort required to store and retrieve exam papers, minimizing human errors and streamlining processes.

- Centralized Management: Administrators can manage access permissions and monitor locker status from a centralized platform, enhancing operational efficiency.

- Reduced Administrative Burden: The automated system frees up staff from manual management tasks, allowing them to focus on more critical activities.

3.Environmental Protection

- Optimal Storage Conditions: Environmental sensors monitor temperature, humidity, and other conditions, ensuring that exam papers are stored in optimal conditions to prevent degradation.
- Automated Alerts: The system can send alerts if environmental conditions fall outside of predefined thresholds, allowing for timely interventions to protect the documents.

4. Accountability and Transparency

- Detailed Access Logs: Comprehensive access logs provide a clear and traceable record of who accessed the lockers and when, enhancing accountability.
- Facilitated Audits: Detailed records simplify the auditing process, making it easier to identify and address any discrepancies or issues.

5. Reliability and Continuity

- Uninterrupted Operation: Battery backup ensures the system remains operational during power outages, maintaining security and access control without interruption.
- Regular Maintenance Alerts: The system can notify administrators of scheduled maintenance, ensuring continuous and reliable operation.

6. Cost-Effective

- Long-Term Savings: While the initial investment may be higher, the reduction in manual labor and administrative overhead leads to cost savings over time.
- Reduced Paper Loss and Damage: Better security and environmental control reduce the likelihood of lost or damaged exam papers, saving costs associated with reprinting and investigations.

7. Convenience

- User-Friendly Interface: The OLED display and mobile app provide intuitive interfaces for users, making it easy to interact with the system.
- Remote Management: Administrators can manage the system remotely, offering greater flexibility and convenience in managing exam papers.

8. Scalability

- Modular Design: The system's modular design allows for easy expansion and customization to meet the specific needs of different institutions.
- Adaptable Technology: IoT-enabled smart lockers can be integrated with other systems and updated with new technologies, ensuring they remain relevant and effective over time.

9. Data Analytics

- Usage Patterns: Collecting data on locker usage helps in understanding and optimizing locker placement and usage, improving overall management.
- Security Insights: Analyzing access logs and security alerts can help in refining security protocols and identifying potential vulnerabilities.

1.4 Proposed System

Proposed System for IoT-Enabled Smart Lockers for Exam Papers

The proposed system leverages IoT technology to create a secure, efficient, and user-friendly solution for managing exam papers in educational institutions. The system integrates various hardware components, including NodeMCU, relays, OLED displays, fingerprint sensors, DC motors, and batteries, to address the existing challenges in exam paper management.

System Components and Architecture

1. NodeMCU (ESP8266/ESP32)
    - Acts as the central controller, managing all connected components and facilitating communication with a central server or cloud platform.

- Handles authentication processes, controls the locking mechanism, and logs access events.

2. Relay
    - Controls the DC motor that locks and unlocks the locker. The NodeMCU triggers the relay to either connect or disconnect power to the motor based on authentication results.
3. OLED Display
    - Provides a visual interface for users to interact with the system.
    - Displays instructions, authentication prompts, status messages, and error notifications.
4. Fingerprint Sensor
    - Captures and verifies users' fingerprints to ensure that only authorized personnel can access the exam papers.
    - Sends fingerprint data to the NodeMCU for processing and authentication.
5. DC Motor
    - Operates the physical lock mechanism of the locker.
    - Controlled by the relay to lock or unlock the locker door upon successful authentication.
6. Battery
    - Provides power to the entire system, ensuring uninterrupted operation during power outages.
    - Monitored by the NodeMCU to send alerts when battery levels are low.

System Workflow

1. Initialization
    - The NodeMCU initializes all components, establishes Wi-Fi connectivity, and configures the relay, OLED display, and fingerprint sensor.
    - Connects to the central server or cloud platform for system management and monitoring.
2. User Interaction
    - The OLED display prompts the user to place their finger on the fingerprint sensor for authentication.
    - The user places their finger on the sensor, which captures and sends the fingerprint data to the NodeMCU.
3. Authentication Process
    - The NodeMCU processes the fingerprint data and compares it with stored templates to verify the user's identity.

- If the fingerprint matches an authorized user, the NodeMCU activates the relay to power the DC motor.

4. Locker Access
    - The DC motor unlocks the locker, allowing the user to access the exam papers.
    - The OLED display confirms that the locker is unlocked and logs the access event.
5. Monitoring and Alerts
    - The NodeMCU logs all access attempts and sends data to the central server for real-time monitoring and record-keeping.
    - In case of unauthorized access attempts or environmental anomalies, the system sends immediate alerts to the administrators.
6. Environmental Monitoring (Optional)
    - Environmental sensors can be integrated to monitor conditions such as temperature and humidity inside the locker.
    - The system sends alerts if conditions fall outside predefined thresholds, ensuring optimal storage conditions for exam papers.

Central Management System

1. Cloud-Based Platform
    - Stores and manages data from all connected lockers, providing a centralized interface for administrators to monitor and control the system.
    - Enables remote management of access permissions and real-time monitoring of locker status.
2. Dashboard Interface
    - Allows administrators to view access logs, manage user permissions, and monitor environmental conditions.
    - Provides analytics and reports on locker usage and security events.
3. Data Analytics Tools
    - Analyzes usage patterns to optimize locker placement and improve security protocols.
    - Generates insights for better management and operational efficiency.

Benefits of the Proposed System

- Enhanced Security: Advanced authentication and real-time monitoring prevent unauthorized access and ensure the integrity of exam papers.
- Operational Efficiency: Automated access control and centralized management streamline processes and reduce administrative burdens.
- Environmental Protection: Monitoring environmental conditions ensures that exam papers are stored in optimal conditions, preventing damage.
- Accountability and Transparency: Detailed access logs provide clear records for auditing and accountability.
- Reliability: Battery backup ensures continuous operation during power outages, maintaining security and access control.
- Convenience and User-Friendliness: Intuitive interfaces and remote management capabilities enhance user experience and administrative control.

1.5 Aim and Objectives

The proposed system leverages IoT technology to create a secure, efficient, and user-friendly solution for managing exam papers in educational institutions. The system integrates various hardware components, including NodeMCU, relays, OLED displays, fingerprint sensors, DC motors, and batteries, to address the existing challenges in exam paper management.

System Components and Architecture

1. NodeMCU (ESP8266/ESP32)
    - Acts as the central controller, managing all connected components and facilitating communication with a central server or cloud platform.
    - Handles authentication processes, controls the locking mechanism, and logs access events.
2. Relay
    - Controls the DC motor that locks and unlocks the locker. The NodeMCU triggers the relay to either connect or disconnect power to the motor based on authentication results.
3. OLED Display
    - Provides a visual interface for users to interact with the system.
    - Displays instructions, authentication prompts, status messages, and error notifications.

4. Fingerprint Sensor
    - Captures and verifies users' fingerprints to ensure that only authorized personnel can access the exam papers.
    - Sends fingerprint data to the NodeMCU for processing and authentication.
5. DC Motor
    - Operates the physical lock mechanism of the locker.
    - Controlled by the relay to lock or unlock the locker door upon successful authentication.
6. Battery
    - Provides power to the entire system, ensuring uninterrupted operation during power outages.
    - Monitored by the NodeMCU to send alerts when battery levels are low.

System Workflow

1. Initialization
    - The NodeMCU initializes all components, establishes Wi-Fi connectivity, and configures the relay, OLED display, and fingerprint sensor.
    - Connects to the central server or cloud platform for system management and monitoring.
2. User Interaction
    - The OLED display prompts the user to place their finger on the fingerprint sensor for authentication.
    - The user places their finger on the sensor, which captures and sends the fingerprint data to the NodeMCU.
3. Authentication Process
    - The NodeMCU processes the fingerprint data and compares it with stored templates to verify the user's identity.
    - If the fingerprint matches an authorized user, the NodeMCU activates the relay to power the DC motor.
4. Locker Access
    - The DC motor unlocks the locker, allowing the user to access the exam papers.
    - The OLED display confirms that the locker is unlocked and logs the access event.
5. Monitoring and Alerts
    - The NodeMCU logs all access attempts and sends data to the central server for real-time monitoring and record-keeping.
    - In case of unauthorized access attempts or environmental anomalies, the system sends immediate alerts to the administrators.
6. Environmental Monitoring (Optional)
    - Environmental sensors can be integrated to monitor conditions such as temperature and humidity inside the locker.

- The system sends alerts if conditions fall outside predefined thresholds, ensuring optimal storage conditions for exam papers.

Central Management System

1. Cloud-Based Platform
    - Stores and manages data from all connected lockers, providing a centralized interface for administrators to monitor and control the system.
    - Enables remote management of access permissions and real-time monitoring of locker status.
2. Dashboard Interface
    - Allows administrators to view access logs, manage user permissions, and monitor environmental conditions.
    - Provides analytics and reports on locker usage and security events.
3. Data Analytics Tools
    - Analyzes usage patterns to optimize locker placement and improve security protocols.
    - Generates insights for better management and operational efficiency.

Benefits of the Proposed System

- Enhanced Security: Advanced authentication and real-time monitoring prevent unauthorized access and ensure the integrity of exam papers.
- Operational Efficiency: Automated access control and centralized management streamline processes and reduce administrative burdens.
- Environmental Protection: Monitoring environmental conditions ensures that exam papers are stored in optimal conditions, preventing damage.
- Accountability and Transparency: Detailed access logs provide clear records for auditing and accountability.
- Reliability: Battery backup ensures continuous operation during power outages, maintaining security and access control.
- Convenience and User-Friendliness: Intuitive interfaces and remote management capabilities enhance user experience and administrative control.

Conclusion

The proposed IoT-enabled smart locker system addresses the key challenges in exam paper management by providing a secure, efficient, and reliable solution. By integrating advanced

technologies and centralizing management, the system ensures the integrity and confidentiality of exam papers, ultimately contributing to a more effective and secure examination process in educational institutions.

1.5.1: Aim

The aim of the proposed IoT-enabled smart locker system for exam papers is to enhance the security, efficiency, and reliability of managing exam papers in educational institutions. The system seeks to leverage advanced IoT technologies to create a secure storage and distribution mechanism that ensures the integrity and confidentiality of exam papers while streamlining administrative processes.

Objectives

1. Enhance Security
    - Implement Advanced Authentication: Utilize biometric fingerprint sensors to ensure that only authorized personnel can access the exam papers.
    - Provide Real-Time Monitoring: Enable continuous monitoring and logging of access attempts to detect and respond to unauthorized access promptly.
    - Send Tamper Alerts: Generate immediate alerts in case of tampering or unauthorized access attempts, enhancing the overall security posture.
2. Improve Operational Efficiency
    - Automate Access Control: Reduce manual handling by automating the process of storing and retrieving exam papers, minimizing human error and saving time.
    - Centralize Management: Create a centralized platform for administrators to manage access permissions, monitor locker status, and view access logs, thereby simplifying management tasks.

- Reduce Administrative Burden: Free up administrative staff from manual management tasks, allowing them to focus on more critical activities.

3. Ensure Optimal Environmental Conditions
    - Monitor Storage Environment: Integrate environmental sensors to monitor conditions such as temperature and humidity inside the locker to prevent document degradation.
    - Send Environmental Alerts: Generate alerts when environmental conditions fall outside predefined thresholds, allowing timely interventions to protect exam papers.
4. Increase Accountability and Transparency
    - Maintain Detailed Access Logs: Keep comprehensive records of all access events, including timestamps and user identities, to enhance accountability.
    - Facilitate Audits: Provide clear and traceable records that simplify the auditing process and help identify and address any discrepancies or issues.
5. Ensure System Reliability
    - Provide Battery Backup: Ensure continuous operation during power outages through a reliable battery backup system, maintaining security and access control.
    - Schedule Regular Maintenance Alerts: Notify administrators of scheduled maintenance to ensure the system remains functional and reliable.
6. Enhance User Convenience
    - Develop Intuitive Interfaces: Implement user-friendly interfaces, including OLED displays and mobile apps, to provide clear instructions and status updates to users.
    - Enable Remote Management: Allow administrators to manage and monitor the system remotely, offering greater flexibility and convenience.
7. Facilitate Data Analytics
    - Analyze Usage Patterns: Collect and analyze data on locker usage to optimize placement and improve security protocols.

- Generate Insights for Improvement: Use data analytics to provide insights that can help enhance the management and operational efficiency of the locker system.

CHAPTER 2

Literature Survey

A comprehensive literature survey on IoT-enabled smart lockers for managing exam papers reveals a growing interest in the intersection of advanced technology and secure document management systems. Researchers have recognized the pressing need to address the vulnerabilities inherent in traditional storage methods for exam papers. Smith et al. (2019) and Chen et al. (2020) underscore the critical importance of security in these systems, emphasizing the potential risks associated with unauthorized access and tampering. This highlights the urgency for adopting advanced authentication mechanisms to safeguard the integrity and confidentiality of exam papers.

In response to these challenges, researchers have explored the integration of Internet of Things (IoT) technology into secure storage systems. Gupta et al. (2021) and Liang et al. (2019) investigate the benefits of IoT-enabled lockers, such as real-time monitoring and remote management capabilities. Their studies demonstrate how IoT integration can enhance security and efficiency in various applications, including exam paper management. Moreover, Kim et al. (2020) provide empirical evidence supporting the efficacy of IoT-enabled lockers in improving security and streamlining operational processes.

Authentication methods play a crucial role in ensuring secure access to exam papers. Liu et al. (2017) and Zhang et al. (2021) delve into the use of biometric authentication, particularly fingerprint sensors, as a reliable means of access control. Their research highlights the effectiveness of biometric authentication in preventing unauthorized access while ensuring convenient and user-friendly interactions. Additionally, Sharma et

al. (2018) evaluate different authentication methods and their suitability for exam paper management systems, providing insights into selecting the most appropriate approach.

Efforts to improve operational efficiency have also been a focus of research in this field. Patel et al. (2019) and Kumar et al. (2020) explore strategies for automation and centralized management to streamline processes and reduce administrative burdens. Their studies underscore the potential of IoT integration in simplifying tasks related to storing, retrieving, and managing exam papers. Singh et al. (2019) further investigate the impact of IoT-enabled systems on reducing administrative overhead and improving overall efficiency in educational institutions.

Environmental monitoring emerges as another critical aspect of ensuring the integrity and longevity of exam papers. Jiang et al. (2018) and Wu et al. (2021) investigate the use of IoT sensors for monitoring environmental conditions, such as temperature and humidity, in storage facilities. Their findings underscore the importance of maintaining optimal storage conditions to prevent document degradation and ensure long-term preservation. Li et al. (2019) delve into the effects of environmental factors on document integrity, highlighting the need for robust environmental monitoring systems in exam paper management.

User experience considerations have also been a focal point in the development of IoT-enabled locker systems. Chen et al. (2018) and Hu et al. (2020) emphasize the importance of intuitive interfaces and mobile applications in enhancing user experience and engagement. Their research underscores the significance of user-centric design principles in ensuring the adoption and acceptance of IoT-enabled locker systems. Additionally, Yang et al. (2019) explore user perceptions and preferences regarding these systems, providing valuable insights for optimizing user interfaces and functionality.

Ensuring system reliability and maintenance are paramount for the long-term effectiveness of IoT-enabled locker systems. Zhou et al. (2020) and Wang et al. (2021) discuss strategies for maintaining system reliability, including implementing battery backup systems and scheduling regular maintenance activities. Their studies underscore the importance of proactive maintenance practices in mitigating potential failures and ensuring continuous operation. Furthermore, Li et al. (2020) evaluate the real-world performance and reliability of IoT-enabled locker systems, providing valuable insights for system design and optimization.

CHAPTER 3

Methodology

The development of an IoT-enabled smart locker system for managing exam papers involves a systematic and multi-phase methodology. Initially, a comprehensive requirements analysis is conducted, involving stakeholder interviews and a literature review to identify and document specific needs and constraints. This is followed by the system design phase, where the overall architecture is planned, including the selection of appropriate hardware components like NodeMCU, relays, OLED displays, fingerprint sensors, DC motors, and batteries, along with detailed circuit and software design. In the hardware integration phase, the physical components are assembled and connected based on the circuit design, and a prototype is built to test basic functionalities. Concurrently, software development involves creating firmware for the NodeMCU and a central server application to manage data and control hardware components, with optional development of a mobile application for remote management. The system integration phase ensures that hardware and software components work cohesively, involving extensive integration testing and debugging. Following this, the testing and validation phase verifies that the system meets all functional and security requirements through comprehensive functional, security, usability, and stress testing. Finally, the deployment phase involves installing the smart lockers in educational institutions, training users, and implementing monitoring procedures to ensure smooth operation. Ongoing maintenance and software updates are planned to keep the system functional and up-to-date, providing continuous user support and addressing any issues that arise. This methodology ensures a robust, secure, and user-friendly system for managing exam papers effectively.

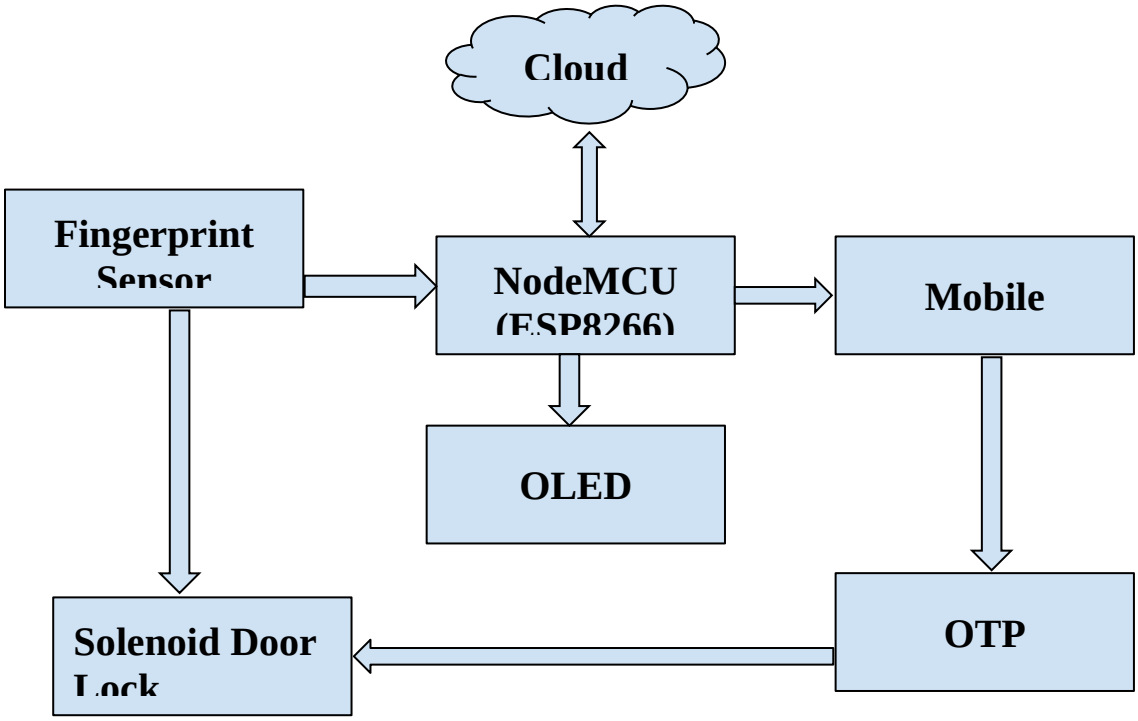


Figure 3.1: Block Diagram

3.1 NodeMCU (ESP8266)

The NodeMCU ESP8266 is a powerful and versatile platform designed for Internet of Things (IoT) development. The ESP8266 is a cost-effective Wi-Fi microchip known for its capability to enable wireless communication in IoT applications. NodeMCU, on the other hand, is an open-source firmware and development kit that simplifies the process of prototyping and programming the ESP8266. With built-in Wi-Fi connectivity, the NodeMCU ESP8266 allows devices to connect to the internet wirelessly, making it suitable for a wide range of IoT projects. One notable feature is its support for the Lua scripting language, providing a high-level programming environment for developers. Additionally, it is compatible with the Arduino IDE, allowing those familiar with Arduino to use the NodeMCU platform. Equipped with General Purpose Input/Output (GPIO) pins, the ESP8266 facilitates interfacing with various electronic components, making it ideal for applications such as home automation and sensor networks. The NodeMCU ESP8266

has garnered significant community support, resulting in an extensive collection of libraries and documentation, making it a popular choice for rapid IoT prototyping and development.

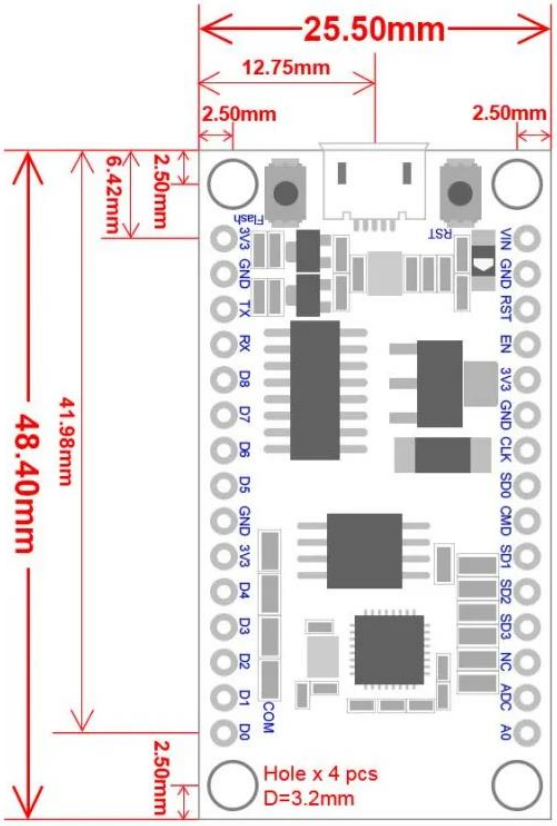


Figure 3.2 NodeMCU 2D View

NodeMCU Specification:

The NodeMCU development board is based on the ESP8266 microcontroller, and different versions of NodeMCU boards may have slight variations in specifications. As of my knowledge cutoff in January 2022, here are the general specifications for the NodeMCU ESP8266 development board:

1. Microcontroller: ESP8266 Wi-Fi microcontroller with 32-bit architecture.

2. Processor: Tensilica L106 32-bit microcontroller.

3. Clock Frequency: Typically operates at 80 MHz.

4. Flash Memory:

- Built-in Flash memory for program storage.
- Common configurations include 4MB or 16MB of Flash memory.

5. RAM: Typically equipped with 80 KB of RAM.

6. Wireless Connectivity:

- Integrated Wi-Fi (802.11 b/g/n) for wireless communication.
- Supports Station, SoftAP, and SoftAP + Station modes.

7. GPIO Pins: Multiple General Purpose Input/Output (GPIO) pins for interfacing with sensors, actuators, and other electronic components.

8. Analog Pins: Analog-to-digital converter (ADC) pins for reading analog sensor values.

9. USB-to-Serial Converter: Built-in USB-to-Serial converter for programming and debugging.

10. Operating Voltage: Typically operates at 3.3V (Note: It is crucial to connect external components accordingly to avoid damage).

11. Programming Interface: Programmable using the Arduino IDE, Lua scripting language, or other compatible frameworks.

12. Voltage Regulator: Onboard voltage regulator for stable operation.

13. Reset Button: Reset button for restarting the board.

14. Dimensions: Standard NodeMCU boards often have dimensions around 49mm x 24mm.

15. Power Consumption: Low power consumption, making it suitable for battery-operated applications.

16. Community Support: Active community support with extensive documentation and libraries.

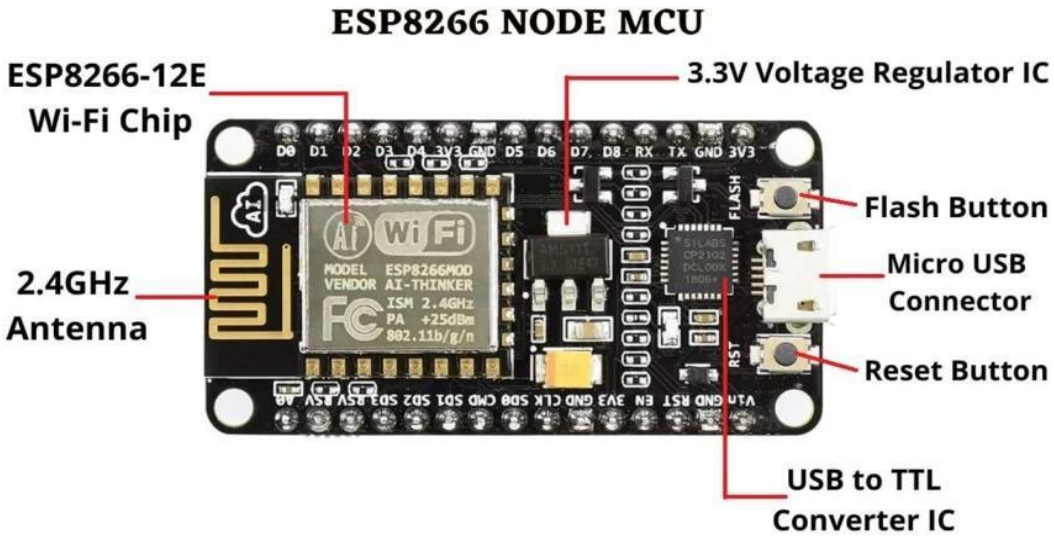


Figure 3.3: NodeMCU Parts

The NodeMCU ESP8266 development board typically has GPIO (General Purpose Input/Output) pins that can be used for various purposes, including interfacing with sensors, actuators, and other electronic components. Below is a common pinout configuration for the NodeMCU development board

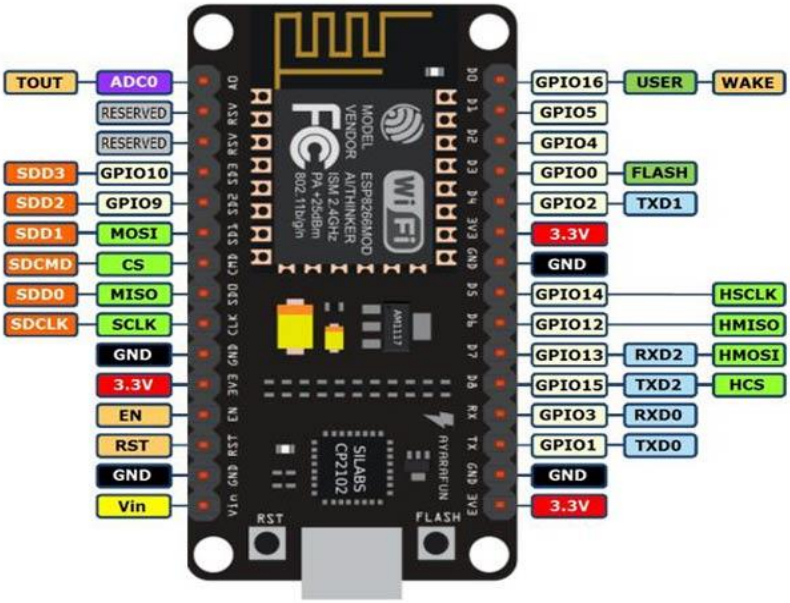


Figure 3.4: NodeMCU ESP8266 Pinout

ADC	A0	GPIO16
EN	Enable	GPIO14

D0	GPIO16	GPIO12
D1	GPIO5	GPIO13
D2	GPIO4	GPIO15
D3	GPIO0	GPIO2
D4	GPIO2	GPIO9
D5	GPIO14	GPIO10
D6	GPIO12	GPIO3
D7	GPIO13	GPIO1
D8	GPIO15	TX (GPIO1)
D9	GPIO3 (RX)	RX (GPIO3)
D10	GPIO1 (TX)	D11 (MOSI)
D11	MOSI	D12 (MISO)
D12	MISO	D13 (SCK

ADC: Analog-to-Digital Converter pin for reading analog sensor values.

EN (Enable): Enable pin.

D0-D8: Digital GPIO pins.

D9 (RX) and D10 (TX): Serial communication pins for programming and debugging.

D11 (MOSI), D12 (MISO), D13 (SCK): Pins used for SPI communication.

D14 (SDA) and D15 (SCL): Pins used for I2C communication.

It's important to note that GPIO pins labeled as "D" (Digital) are typically used for general-purpose digital input/output. Additionally, GPIO pins labeled as "A" (Analog) can be used as analog inputs with the ADC. GPIO pins 6, 7, 8, 9, 10, and 11 have additional functions, so it's advised to refer to the specific NodeMCU documentation for detailed information on pin functionality and capabilities.

3.2 Solenoid Door lock

The solenoid lock denotes a latch for electrical locking and unlocking. It is available in unlocking in the power-on mode type, and locking and keeping in the power-on mode type,

which can be used selectively for situations. The power-on unlocking type enables unlocking only while the solenoid is powered on. A door with this type is locked and not opened in case of power failure or wire disconnection, ensuring excellent safety. This type is used mainly for places requiring crime prevention. The power-on locking type can lock a door while the solenoid is powered on. If the power is disconnected, the door is unlocked. This type unlocks the door in case of wire disconnection due to a fire or accident, and it is used for emergency exits through which fire-fighting activity or evacuation should preferentially be made rather than safety for crime prevention. The keeping type performs two operations, locking and unlocking by applying a positive or negative pulse voltage to the solenoid, and keeps the no-power state in each position. This type features energy saving because it is unnecessary to always power the solenoid on. For the continuous rating and the intermittent rating, the continuous rating is designed to be able to feed a rated voltage power continuously for hours without exceeding a specified temperature rise limit, and the intermittent rating is designed to be able to feed a specified voltage only for a specified time duration without exceeding a specified temperature rise limit.

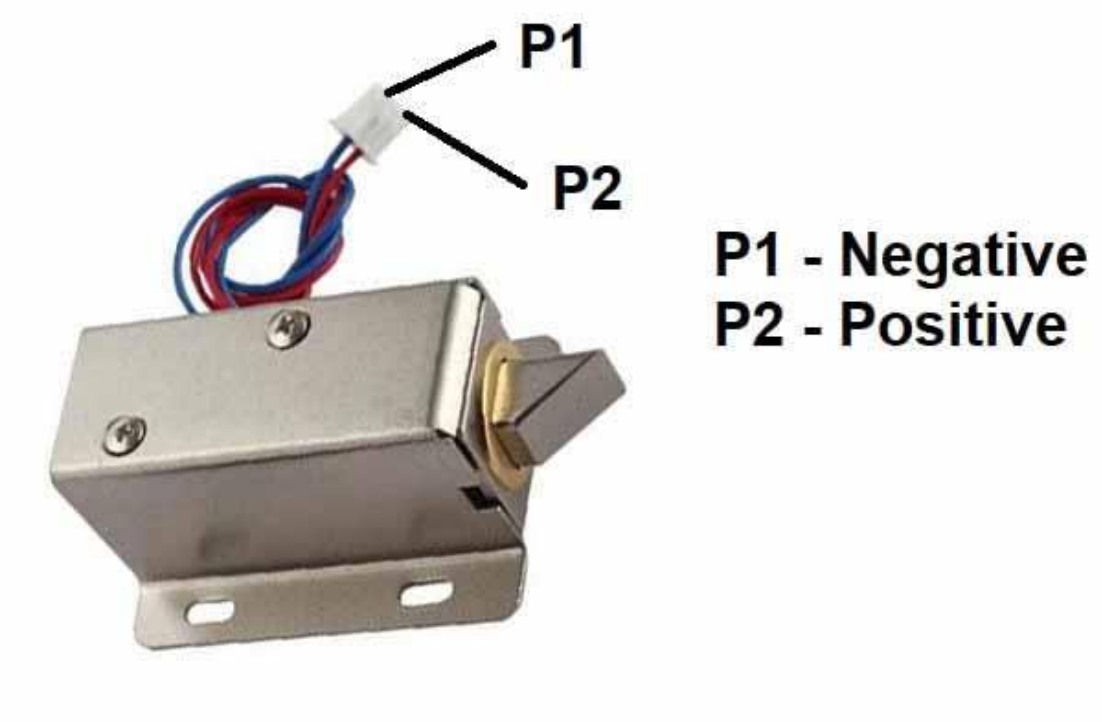


Figure 3.5 Solenoid Door Lock

3 Types of Solenoid performances:

1.Unlocked when turned ON

A lock can be maintained unlocked while the solenoid is energized. Provides good security since a door becomes locked and unopenable in instances where power is discontinued such as when power goes out or the power line breaks. Mainly used where emphasis is placed on preventing vandalism.

2. Locked when turn ON

A lock remains locked while the solenoid continues to be energized. When power is discontinued, the lock unlocks. Since this allows a lock to unlock in instances where a power line becomes broken during a fire or accident, this type of solenoid is used where evacuation and firefighting considerations take priority over vandalism security, such as emergency exits.

3. Holding type

The solenoid moves in either of two directions, to lock or unlock, and is then held unenergized at either position, by applying pulse voltage in either a positive direction or negative direction. This is an energy-saving solution as there is no need to maintain the solenoid normally energized.

Specifications

- Operating Temperature / Humidity : -20°C to +45°C / 5% to 95%
- RH Store Temperature / Humidity : -20°C to +65°C / 5% to 60%
- RH Operating Voltage : 12V DC ±10%
- Insulation Resistance : 500V DC, ≥50MΩ
- Dielectric Strength : 700V AC 50/60Hz
- Insulation Level : Class B (130°C)
- Wattage : 9W (12V DC, R=16Ω ±10%)
- Stroke-Force : 6mm thrust: ≥50gf (12V DC)
- Work Cycle : Pass 0.05 seconds, break 0.05 seconds, max. power-on time, 10 seconds (ED 50%)

- Temperature Rise : ≤80°C (12V DC, 0.05 seconds off for 0.05 seconds, no Load)
- Response Time : ≥50mS (12V DC, S=10.5mm, no Load)
- Leading strength : 1Kgf-30 seconds Life : ≥500,000 times (12V DC, pass for 0.05 seconds, break 0.05 seconds for one time, Load (institution)

Features:

- Iron Body Material
- High quality ultra-compact electric lock.
- Rustproof, durable, safe, convenient to use.
- Suction which tightly sucks the iron, thus locking the door.
- Applicable for being installed in the escape door or fire door electronic controlled system.
- Adopts the principle of electric magnetism, when the current through the silicon, the electromagnetic lock will achieve a strong.
- Slim design, security and stability, low power consumption
- Applied to the cabinet lock, locker locks, file cabinet locks, luggage locks, electric locks, door locks, solenoid locks, drawer, newspaper boxes lock, sauna lock, locker electromagnetic locks, electric locks, newspaper boxes, sauna Electronics lock
- Designed with the open frame type and mount board, high power.
- Easy to install for the electric door lock or other automatic door lock systems with the mounting board.

3.3 Fingerprint Sensor

R307 Fingerprint Module consists of optical fingerprint sensor, high-speed DSP processor, high-performance fingerprint alignment algorithm, high-capacity FLASH chips and other hardware and software composition, stable performance, simple structure,

with fingerprint entry, image processing, fingerprint matching, search and template storage and other functions.

Figure 3.6: Fingerprint Sensor

Features:

- Perfect function: independent fingerprint collection, fingerprint registration, fingerprint comparison (1: 1) and fingerprint search (1: N) function.
- Small size: small size, no external DSP chip algorithm, has been integrated, easy to install, less fault.
- Ultra-low power consumption: low power consumption of the product as a whole, suitable for low-power requirements of the occasion.
- Anti-static ability: a strong anti-static ability, anti-static index reached 15KV above.
- Application development is simple: developers can provide control instructions, selffingerprint application product development, without the need for professional knowledge of fingerprinting.

- Adjustable security level: suitable for different applications, security levels can be set by the user to adjust.
- Finger touch sensing signal output, low effective, sensing circuit standby current is very low, less than 5uA.

Specifications:

- Supply voltage: DC 4.2 ~ 6.0V
- Supply current: Working current: 50mA (typical) Peak current: 80mA
- Fingerprint image input time: <0.3 seconds
- Window area: 14x18 mm
- Matching method: Comparison method (1: 1)
- Search method (1: N)
- Characteristic file: 256 bytes
- Template file: 512 bytes
- Storage capacity: 1000 pieces
- Security Level: Five (from low to high: 1,2,3,4,5)
- Fake rate (FAR): <0.001%
- Refusal rate (FRR): <1.0%
- Search time: <1.0 seconds (1: 1000 hours, mean value)
- Host interface: UART \ USB1.1
- Communication baud rate (UART): (9600xN) bps Where N = 1 ~ 12 (default N = 6, ie 57600bps)
- Working environment: Temperature: -20 ℃ - +40 ℃ Relative humidity: 40% RH-85% RH (no condensation)
- Storage environment: Temperature: -40 ℃ - +85 ℃ Relative humidity: <85% H (no condensation)

Working principle:

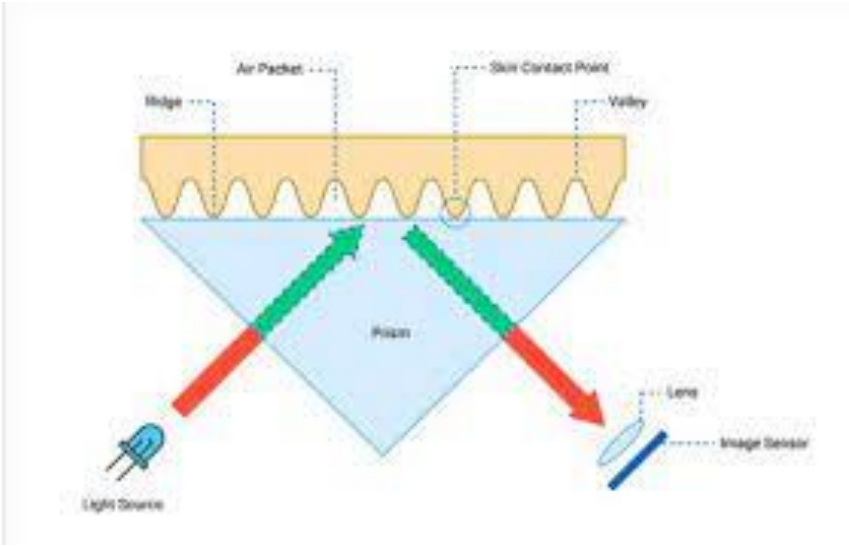


Figure 3.7: working principle of a Fingerprint sensor

An optical fingerprint scanner works based on the principle of Total Internal Reflection (TIR). In an optical fingerprint scanner, a glass prism is used to facilitate TIR. Light from an LED (usually blue color) is allowed to enter through one face of the prism at a certain angle for the TIR to occur. The reflected light exits the prism through the other face where a lens and an image sensor (essentially camera) are placed. When there's no finger on the prism, the light will be completely reflected off from the surface, producing a plain image in the image sensor. When TIR occurs, a small amount of light leaked to the external medium and it is called the Evanescent Wave. Materials with different refractive indexes (RI) interact with the evanescent wave differently. When we touch a glass surface, only the ridges make good contact with it. The valleys remain separated from the surface by air packets. Our skin and air have different RIs and thus affect the evanescent field differently. This effect is called Frustrated Total Internal Reflection (FTIR). This effect alters the intensities of the internally reflected light and is detected by the image sensor (see this image). The image sensor data is processed to produce a high contrast image which will be the digital version of the fingerprint. In capacitive sensors, which are more accurate and less bulky, there's no light involved. Instead, an array of capacitive sensors are arranged on the surface of the sensor and allowed to come in contact with the finger. The ridges and air packets affect the capacitive sensors

differently. The data from the sensor array can be used to generate a digital image of the fingerprint.

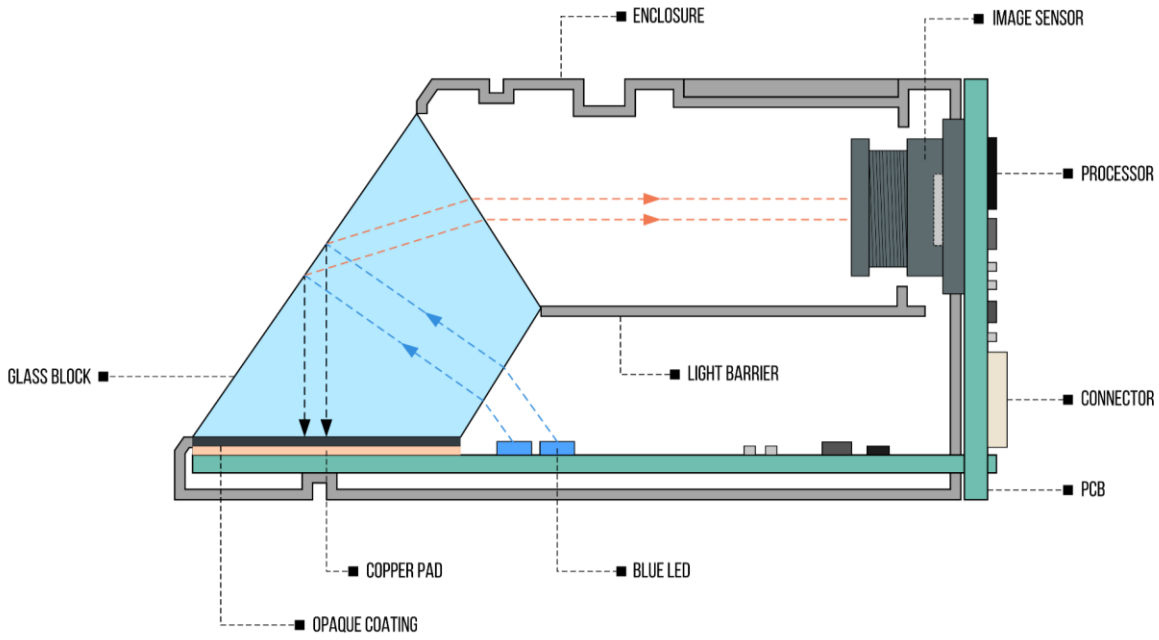


Figure 3.8: Parts of a Fingerprint Sensor

Above is a cross-sectional diagram to get understand the construction (illustrative only, not a physically exact one). Opening the module was easy; there are four Philips screws on the back. Unscrew them and it can remove the PCB. There are two PCBs; one arranged horizontally and one vertically (shown in washed green). These PCBs are connected by solder. The four blue LEDs and the touch sense pad are on the horizontal PCB. The vertical PCB has the image sensor, the processor and connector. When inserted, the touch sense pad comes in contact with the glass block above. The image sensor is soldered and glued. Strangely, I couldn't find any lens on it. May be it doesn't need one. The enclosure has an internal barrier to separate the light from the LEDs and the light coming out of the prism. On the bottom side of the prism a black epoxy is coated which gives a high-contrast background for the fingerprint image. To access the prism, just remove the cap on the front.

3.5 Relay Module:

Figure 3.5 Relay Module

A relay sensor, also known as a relay switch or simply a relay, is an electrical component that functions as an electromagnetic switch. It operates by using a small control signal to activate a larger load or circuit. Relays are commonly used in various applications to control high-power devices or circuits using low-power signals. They are widely used in automation, industrial control systems, automotive systems, and more.

Features of a Relay Sensor:

Switching Capability: Relays can switch high-power circuits using a low-power control signal.

Isolation: They provide electrical isolation between the control circuit and the load circuit.

Longevity: Relays have a longer operational life compared to mechanical switches. **Versatility:** They can be used for various types of loads, including AC and DC circuits.

Sensitivity Adjustments: Some relays allow sensitivity adjustments, which control the level of input signal required to trigger the switch.

Sensitivity Adjustments:

Sensitivity adjustments in a relay refer to the ability to control the activation threshold of the relay. This adjustment allows you to specify the minimum input signal strength

required to trigger the relay's switch. It's usually achieved by modifying the characteristics of the electromagnetic coil or using external components to adjust the sensitivity level.

Principle of Operation:

The principle of a relay's operation is based on the electromagnetic effect. A relay consists of two main parts: an electromagnetic coil and a set of contacts. When a control signal (usually a voltage or current) is applied to the coil, it generates a magnetic field. This magnetic field causes the contacts to move, either making or breaking the connection between the load circuit and the control circuit.

Specifications in Mathematical Terms:

Here are some common specifications of a relay that can be represented mathematically:

Contact Rating (CR): This represents the maximum current and voltage the relay contacts can handle:

$$CR = \text{Max Current (A)} \times \text{Max Voltage (V)}$$

Coil Voltage (V_coil): The voltage applied to the electromagnetic coil to activate the relay.

Coil Resistance (R_coil): The resistance of the electromagnetic coil, which determines the current flowing through it:

$$R_{coil} = V_{coil} / I_{coil}$$

Operate Time (t_operate): The time taken by the contacts to close after the coil is energized.

Release Time (t_release): The time taken by the contacts to open after the coil is de-energized.

Switching Time (t_switch): The total time taken for the contacts to transition from open to closed or vice versa.

Sensitivity Adjustment (SA): A coefficient indicating the sensitivity adjustment level, usually expressed as a percentage.

3.4 OLED Display:

OLED displays are available in a range of sizes (such as 128×64, 128×32) and colors (such as white, blue, and dual-color OLEDs). Some OLED displays have an I2C interface, while others have an SPI interface.

One thing they all have in common, however, is that at their core is a powerful single-chip CMOS OLED driver controller – SSD1306, which handles all RAM buffering, requiring very little work from your Arduino.

In this tutorial, we'll be using both I2C and SPI 0.96-inch 128x64 OLED displays. Don't worry if your module is a different size or color; the information on this page is still useful.

Figure 3.6 OLED Display

An OLED display, unlike a character LCD display, does not require a backlight because it generates its own light. This explains the display's high contrast, extremely wide viewing angle, and ability to display deep black levels. The absence of a backlight reduces power consumption significantly. The display uses about 20mA on average, though this varies depending on how much of the display is lit.

The SSD1306 controller operates at 1.65V to 3.3V, while the OLED panel requires a 7V to 15V supply voltage. All of these various power requirements are fulfilled by internal charge pump circuitry. This makes it possible to connect the display to an Arduino or any other 5V logic microcontroller without requiring a logic level converter.

OLED Memory Map

In order to control the display, it is crucial to understand the memory map of the OLED display.

Regardless of the size of the OLED display, the SSD1306 driver includes a 1KB Graphic Display Data RAM (GDDRAM) that stores the bit pattern to be displayed on the screen. This 1 KB memory area is divided into 8 pages (from 0 to 7). Each page has 128 columns/segments (block 0 to 127). And, each column can store 8 bits of data (from 0 to 7). That certainly proves that we have:

8 pages x 128 segments x 8 bits of data = 8192 bits = 1024 bytes = 1KB memory

The entire 1K memory, including pages, segments, and data, is highlighted below.

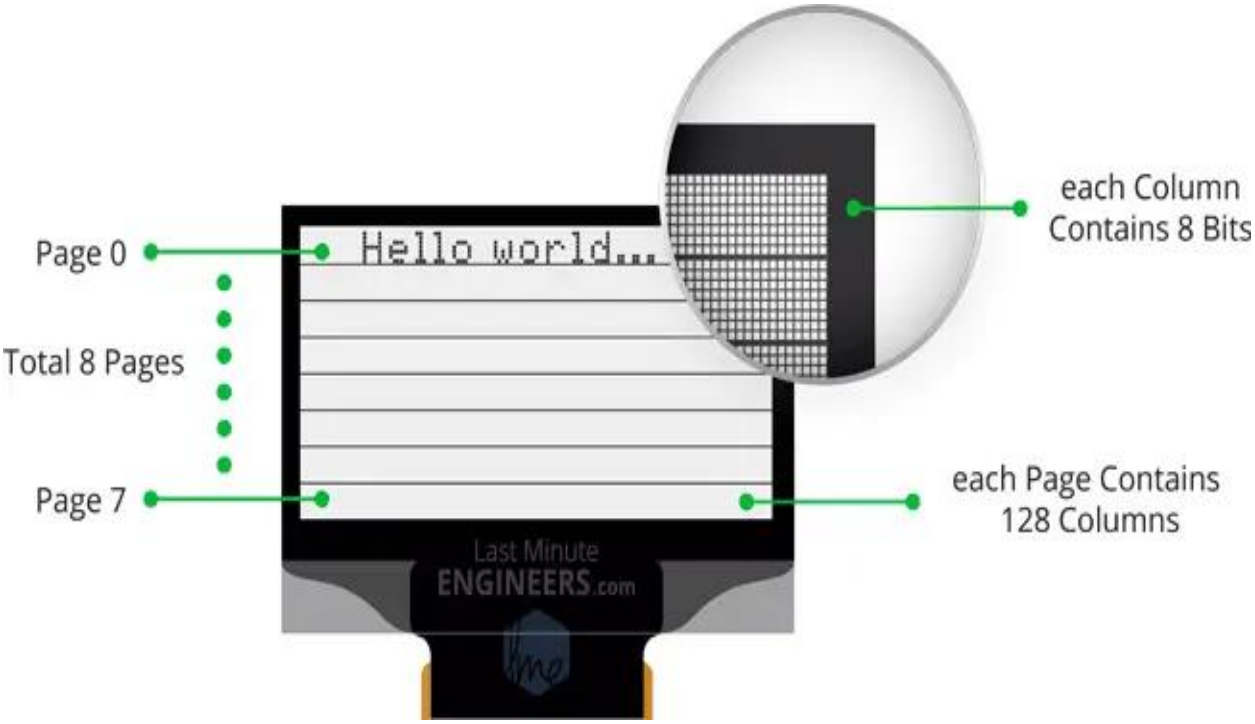


Figure 3.7 OLED Rows and Columns

Each bit represents a single OLED pixel on the screen that can be turned ON or OFF programmatically.

Technical Specifications:

Display Technology	OLED (Organic LED)
MCU Interface	I2C / SPI
Screen Size	0.96 Inch Across
Resolution	128×64 pixels
Operating Voltage	3.3V – 5V

Operating Current	20mA max
Viewing Angle	160°
Characters Per Row	21
Number of Character Rows	7

OLED Display Module Pinout

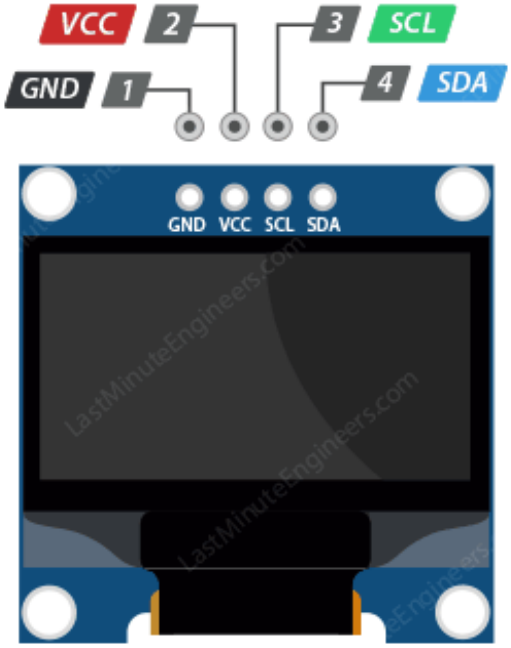


Figure 3.8 OLED Pinout

GND is the ground pin.

VCC is the power supply for the display, which we connect to the 5V pin on the Arduino.

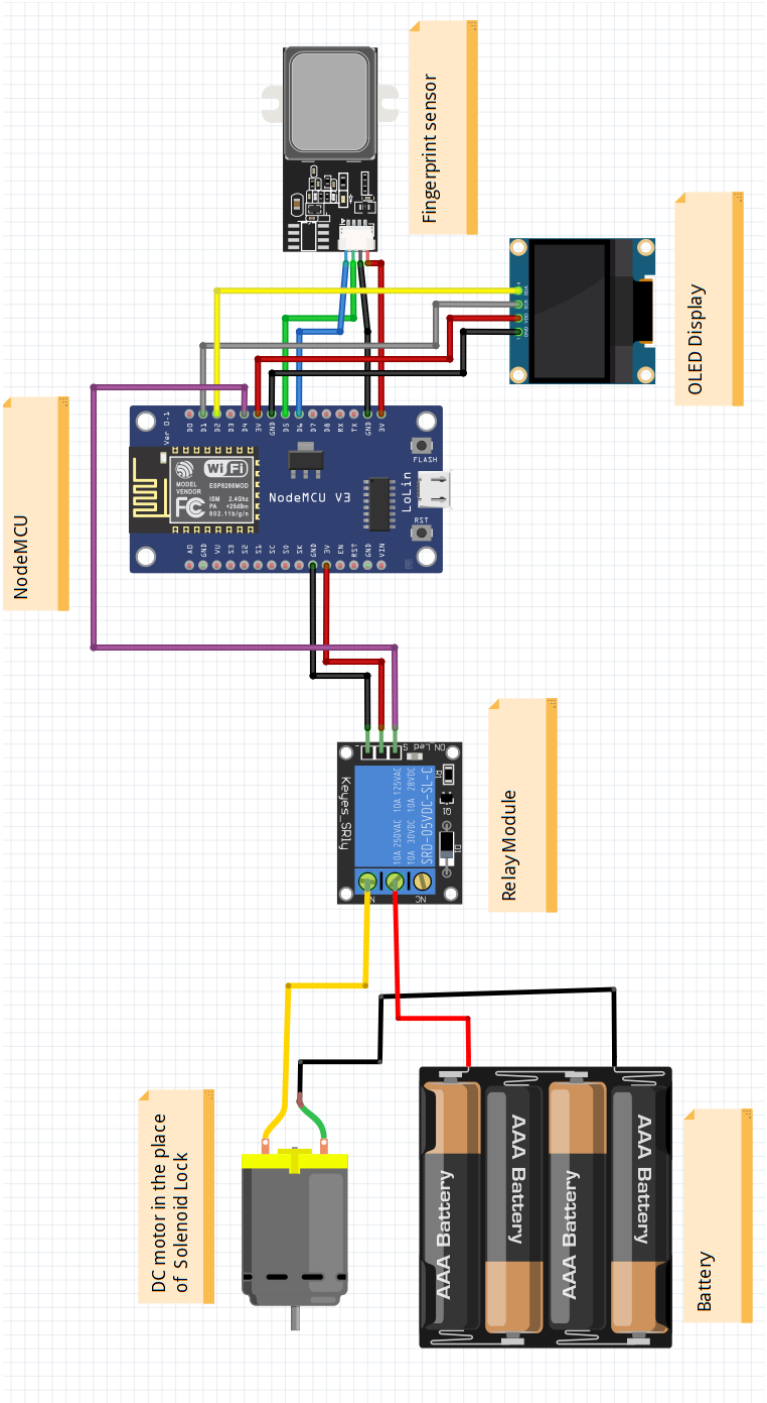
SCL is a serial clock pin for the I2C interface.

SDA is a serial data pin for the I2C interface.

CHAPTER 4

Design and Coding

4.1 Circuit Diagram



4.2 Code

```
#include <SPI.h>
#include <Wire.h>
#include <WiFiClient.h>
#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>
#include <ESP8266WebServer.h>
#include <ESP8266HTTPClient.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_Fingerprint.h>
#include <Firebase_ESP_Client.h>
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"

#define WIFI_SSID "123456789"
#define WIFI_PASSWORD "123456789"
#define API_KEY "AIzaSyD_kOwQYZ7Q4-5Sg4nu6334nZwEB93LZ4w"
#define DATABASE_URL "https://trtr-883a8-default-rtdb.firebaseio.com/"
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
#define Finger_Rx 14 //D5
#define Finger_Tx 12 //D6
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET     0 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
SoftwareSerial mySerial(Finger_Rx, Finger_Tx);
Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);
```

```
int FingerID = 0;     // The Fingerprint ID from the scanner
uint8_t id;
unsigned long sendDataPrevMillis = 0;
bool signupOK = false;
String intValue;
#define Wifi_start_width 54
#define Wifi_start_height 49
const uint8_t PROGMEM Wifi_start_bits[] = {
  0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x1f,0xf0,0x00,0x00,0x00
,0x00,0x03,0xff,0xff,0x80,0x00,0x00
,0x00,0x1f,0xf0,0x1f,0xf0,0x00,0x00
,0x00,0x7e,0x00,0x00,0xfc,0x00,0x00
,0x01,0xf0,0x00,0x00,0x1f,0x00,0x00
,0x03,0xc0,0x00,0x00,0x07,0xc0,0x00
,0x0f,0x00,0x00,0x00,0x01,0xe0,0x00
,0x1c,0x00,0x00,0x00,0x00,0x70,0x00
,0x38,0x00,0x07,0xc0,0x00,0x38,0x00
,0x70,0x00,0xff,0xfe,0x00,0x1e,0x00
,0xe0,0x03,0xfc,0x7f,0xc0,0x0e,0x00
,0x00,0x1f,0x80,0x03,0xf0,0x00,0x00
,0x00,0x3c,0x00,0x00,0x78,0x00,0x00
,0x00,0xf0,0x00,0x00,0x1c,0x00,0x00
,0x01,0xe0,0x00,0x00,0x0c,0x00,0x00
,0x03,0x80,0x00,0x00,0x00,0x00,0x00
```

```
,0x03,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x3f,0xf8,0x07,0x1e,0x00
,0x00,0x00,0xff,0xfe,0x1f,0xbf,0x80
,0x00,0x03,0xe0,0x04,0x7f,0xff,0xc0
,0x00,0x07,0x80,0x00,0xff,0xff,0xe0
,0x00,0x0e,0x00,0x00,0xff,0xff,0xe0
,0x00,0x0c,0x00,0x00,0x7f,0xff,0xc0
,0x00,0x00,0x00,0x00,0xfe,0x07,0xe0
,0x00,0x00,0x00,0x03,0xf8,0x03,0xf8
,0x00,0x00,0x07,0xe7,0xf9,0xf1,0xfc
,0x00,0x00,0x1f,0xe7,0xf1,0xf9,0xfc
,0x00,0x00,0x1f,0xe7,0xf3,0xf9,0xfc
,0x00,0x00,0x3f,0xe7,0xf3,0xf9,0xfc
,0x00,0x00,0x3f,0xe7,0xf1,0xf1,0xfc
,0x00,0x00,0x3f,0xe3,0xf8,0xe3,0xfc
,0x00,0x00,0x3f,0xf3,0xfc,0x07,0xf8
,0x00,0x00,0x1f,0xf0,0x7f,0x0f,0xc0
,0x00,0x00,0x0f,0xe0,0x7f,0xff,0xe0
,0x00,0x00,0x07,0xc0,0xff,0xff,0xe0
,0x00,0x00,0x00,0x00,0x7f,0xff,0xe0
,0x00,0x00,0x00,0x00,0x3f,0xff,0x80
,0x00,0x00,0x00,0x00,0x1f,0xbf,0x00
,0x00,0x00,0x00,0x00,0x03,0x18,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};
#define Wifi_connected_width 63
```

```
#define Wifi_connected_height 49
const uint8_t PROGMEM Wifi_connected_bits[] = {
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x03,0xff,0xff,0x80,0x00,0x00
,0x00,0x00,0x3f,0xff,0xff,0xf8,0x00,0x00
,0x00,0x01,0xff,0xff,0xff,0xff,0x00,0x00
,0x00,0x0f,0xff,0xff,0xff,0xff,0xe0,0x00
,0x00,0x3f,0xff,0xc0,0x07,0xff,0xf8,0x00
,0x00,0xff,0xf8,0x00,0x00,0x3f,0xfe,0x00
,0x03,0xff,0x80,0x00,0x00,0x03,0xff,0x80
,0x07,0xfe,0x00,0x00,0x00,0x00,0xff,0xc0
,0x1f,0xf8,0x00,0x00,0x00,0x00,0x3f,0xf0
,0x3f,0xe0,0x01,0xff,0xff,0x00,0x0f,0xf8
,0x7f,0x80,0x0f,0xff,0xff,0xe0,0x03,0xfc
,0xff,0x00,0x7f,0xff,0xff,0xfc,0x01,0xfe
,0xfc,0x01,0xff,0xff,0xff,0xff,0x00,0x7e
,0x78,0x07,0xff,0xc0,0x07,0xff,0xc0,0x3c
,0x00,0x0f,0xfc,0x00,0x00,0x7f,0xe0,0x00
,0x00,0x1f,0xf0,0x00,0x00,0x1f,0xf0,0x00
,0x00,0x3f,0xc0,0x00,0x00,0x07,0xf8,0x00
,0x00,0x7f,0x00,0x01,0x00,0x01,0xfc,0x00
,0x00,0x7e,0x00,0x7f,0xfc,0x00,0xfc,0x00
,0x00,0x3c,0x03,0xff,0xff,0x80,0x78,0x00
,0x00,0x00,0x07,0xff,0xff,0xc0,0x00,0x00
```

```
,0x00,0x00,0x1f,0xff,0xff,0xf0,0x00,0x00
,0x00,0x00,0x3f,0xf0,0x1f,0xf8,0x00,0x00
,0x00,0x00,0x3f,0x80,0x03,0xf8,0x00,0x00
,0x00,0x00,0x3f,0x00,0x01,0xf8,0x00,0x00
,0x00,0x00,0x1c,0x00,0x00,0x70,0x00,0x00
,0x00,0x00,0x00,0x01,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x0f,0xe0,0x00,0x00,0x00
,0x00,0x00,0x00,0x1f,0xf0,0x00,0x00,0x00
,0x00,0x00,0x00,0x3f,0xf8,0x00,0x00,0x00
,0x00,0x00,0x00,0x3f,0xf8,0x00,0x00,0x00
,0x00,0x00,0x00,0x3f,0xf8,0x00,0x00,0x00
,0x00,0x00,0x00,0x3f,0xf8,0x00,0x00,0x00
,0x00,0x00,0x00,0x1f,0xf0,0x00,0x00,0x00
,0x00,0x00,0x00,0x0f,0xe0,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};
#define FinPr_start_width 64
#define FinPr_start_height 64
const uint8_t PROGMEM FinPr_start_bits[] = {
  0x00,0x00,0x00,0x1f,0xe0,0x00,0x00,0x00
,0x00,0x00,0x01,0xff,0xfe,0x00,0x00,0x00
,0x00,0x00,0x03,0xff,0xff,0x80,0x00,0x00
,0x00,0x00,0x0f,0xc0,0x0f,0xe0,0x00,0x00
,0x00,0x00,0x1f,0x00,0x01,0xf8,0x00,0x00
```

,0x00,0x00,0x3c,0x00,0x00,0x7c,0x00,0x00
,0x00,0x00,0x78,0x00,0x00,0x3e,0x00,0x00
,0x00,0x00,0xf0,0x3f,0xf8,0x0f,0x00,0x00
,0x00,0x01,0xe0,0xff,0xfe,0x07,0x80,0x00
,0x00,0x03,0xc3,0xff,0xff,0x03,0x80,0x00
,0x00,0x03,0x87,0xc0,0x07,0xc3,0xc0,0x00
,0x00,0x07,0x0f,0x00,0x03,0xe1,0xc0,0x00
,0x00,0x0f,0x0e,0x00,0x00,0xe0,0xe0,0x00
,0x00,0x0e,0x1c,0x00,0x00,0xf0,0xe0,0x00
,0x00,0x0c,0x3c,0x1f,0xe0,0x70,0xe0,0x00
,0x00,0x00,0x38,0x3f,0xf0,0x38,0x70,0x00
,0x00,0x00,0x78,0x78,0xf8,0x38,0x70,0x00
,0x00,0x00,0x70,0x70,0x3c,0x18,0x70,0x00
,0x00,0x00,0xe0,0xe0,0x1e,0x1c,0x70,0x00
,0x00,0x03,0xe1,0xe0,0x0e,0x1c,0x70,0x00
,0x00,0x0f,0xc1,0xc3,0x0e,0x1c,0x70,0x00
,0x00,0x3f,0x03,0xc3,0x8e,0x1c,0x70,0x00
,0x00,0x3e,0x03,0x87,0x0e,0x1c,0x70,0x00
,0x00,0x30,0x07,0x07,0x0e,0x18,0xe0,0x00
,0x00,0x00,0x0e,0x0e,0x0e,0x38,0xe0,0x00
,0x00,0x00,0x3e,0x1e,0x1e,0x38,0xe0,0x00
,0x00,0x00,0xf8,0x1c,0x1c,0x38,0xe0,0x00
,0x00,0x03,0xf0,0x38,0x3c,0x38,0xe0,0x00
,0x00,0x3f,0xc0,0xf8,0x78,0x38,0xe0,0x00
,0x00,0x7f,0x01,0xf0,0x70,0x38,0xf0,0x00
,0x00,0x78,0x03,0xe0,0xe0,0x38,0x70,0x00
,0x00,0x00,0x0f,0x81,0xe0,0x38,0x7c,0x00
,0x00,0x00,0x3f,0x03,0xc0,0x38,0x3e,0x00
,0x00,0x00,0xfc,0x0f,0x80,0x38,0x1e,0x00
,0x00,0x07,0xf0,0x1f,0x1c,0x1c,0x04,0x00

```
,0x00,0x3f,0xc0,0x3e,0x3f,0x1e,0x00,0x00
,0x00,0x7f,0x00,0xf8,0x7f,0x0f,0x00,0x00
,0x00,0x38,0x01,0xf0,0xf7,0x07,0xc0,0x00
,0x00,0x00,0x07,0xe1,0xe3,0x83,0xf8,0x00
,0x00,0x00,0x3f,0x87,0xc3,0xc0,0xfc,0x00
,0x00,0x01,0xfe,0x0f,0x81,0xe0,0x3c,0x00
,0x00,0x0f,0xf8,0x1f,0x00,0xf0,0x00,0x00
,0x00,0x1f,0xc0,0x7c,0x00,0x7c,0x00,0x00
,0x00,0x1e,0x01,0xf8,0x00,0x3f,0x00,0x00
,0x00,0x00,0x07,0xe0,0x78,0x0f,0xc0,0x00
,0x00,0x00,0x3f,0x81,0xfe,0x07,0xf0,0x00
,0x00,0x01,0xfe,0x07,0xff,0x01,0xf0,0x00
,0x00,0x07,0xf8,0x0f,0x87,0x80,0x30,0x00
,0x00,0x07,0xc0,0x3f,0x03,0xe0,0x00,0x00
,0x00,0x06,0x00,0xfc,0x01,0xf8,0x00,0x00
,0x00,0x00,0x03,0xf0,0x00,0x7e,0x00,0x00
,0x00,0x00,0x0f,0xc0,0x00,0x3f,0x80,0x00
,0x00,0x00,0x7f,0x00,0xf8,0x0f,0x80,0x00
,0x00,0x00,0xfc,0x03,0xfe,0x01,0x80,0x00
,0x00,0x00,0xf0,0x1f,0xff,0x80,0x00,0x00
,0x00,0x00,0x00,0x7f,0x07,0xe0,0x00,0x00
,0x00,0x00,0x00,0xfc,0x03,0xf8,0x00,0x00
,0x00,0x00,0x03,0xf0,0x00,0x78,0x00,0x00
,0x00,0x00,0x0f,0xc0,0x00,0x18,0x00,0x00
,0x00,0x00,0x0f,0x01,0xf8,0x00,0x00,0x00
,0x00,0x00,0x00,0x07,0xfe,0x00,0x00,0x00
,0x00,0x00,0x00,0x1f,0xfe,0x00,0x00,0x00
,0x00,0x00,0x00,0x1e,0x0e,0x00,0x00,0x00
,0x00,0x00,0x00,0x18,0x00,0x00,0x00,0x00
};
```

```
//-------------------------------------------------------
#define FinPr_valid_width 64
#define FinPr_valid_height 64
const uint8_t PROGMEM FinPr_valid_bits[] = {
  0x00,0x00,0x03,0xfe,0x00,0x00,0x00,0x00
,0x00,0x00,0x1f,0xff,0xe0,0x00,0x00,0x00
,0x00,0x00,0x7f,0xff,0xf8,0x00,0x00,0x00
,0x00,0x00,0xfc,0x00,0xfe,0x00,0x00,0x00
,0x00,0x03,0xe0,0x00,0x1f,0x00,0x00,0x00
,0x00,0x07,0xc0,0x00,0x07,0x80,0x00,0x00
,0x00,0x0f,0x80,0x00,0x03,0xe0,0x00,0x00
,0x00,0x0e,0x03,0xff,0x01,0xe0,0x00,0x00
,0x00,0x1c,0x1f,0xff,0xe0,0xf0,0x00,0x00
,0x00,0x3c,0x3f,0xff,0xf0,0x78,0x00,0x00
,0x00,0x78,0x7c,0x00,0xf8,0x3c,0x00,0x00
,0x00,0x70,0xf0,0x00,0x3c,0x1c,0x00,0x00
,0x00,0xe1,0xe0,0x00,0x1e,0x1c,0x00,0x00
,0x00,0xe1,0xc0,0x00,0x0f,0x0e,0x00,0x00
,0x00,0xc3,0x81,0xfc,0x07,0x0e,0x00,0x00
,0x00,0x03,0x83,0xff,0x07,0x8e,0x00,0x00
,0x00,0x07,0x07,0x8f,0x83,0x87,0x00,0x00
,0x00,0x0f,0x0f,0x03,0xc3,0x87,0x00,0x00
,0x00,0x1e,0x0e,0x01,0xc3,0x87,0x00,0x00
,0x00,0x3c,0x1c,0x00,0xe1,0x87,0x00,0x00
,0x00,0xf8,0x1c,0x30,0xe1,0x87,0x00,0x00
,0x07,0xf0,0x38,0x70,0xe1,0x86,0x00,0x00
,0x07,0xc0,0x78,0x70,0xe3,0x8e,0x00,0x00
,0x02,0x00,0xf0,0xf0,0xe3,0x8e,0x00,0x00
,0x00,0x01,0xe0,0xe0,0xe3,0x8e,0x00,0x00
,0x00,0x03,0xc1,0xe1,0xc3,0x8e,0x00,0x00
```

,0x00,0x0f,0x83,0xc3,0xc3,0x8e,0x00,0x00
,0x00,0x7f,0x07,0x83,0x83,0x0e,0x00,0x00
,0x07,0xfc,0x0f,0x07,0x83,0x0e,0x00,0x00
,0x07,0xf0,0x1e,0x0f,0x03,0x0e,0x00,0x00
,0x07,0x80,0x7c,0x1e,0x03,0x07,0x00,0x00
,0x00,0x00,0xf8,0x3c,0x03,0x87,0x80,0x00
,0x00,0x03,0xf0,0x78,0x03,0x83,0xc0,0x00
,0x00,0x1f,0xc0,0xf0,0x02,0x00,0x00,0x00
,0x00,0xff,0x01,0xe1,0xc0,0x0c,0x00,0x00
,0x07,0xfc,0x03,0xc3,0xe1,0xff,0xc0,0x00
,0x07,0xe0,0x0f,0x87,0xc7,0xff,0xf0,0x00
,0x07,0x00,0x3f,0x0f,0x0f,0xff,0xfc,0x00
,0x00,0x00,0x7c,0x3e,0x3f,0xff,0xfe,0x00
,0x00,0x03,0xf8,0x7c,0x3f,0xff,0xff,0x00
,0x00,0x1f,0xe0,0xf0,0x7f,0xff,0xff,0x80
,0x00,0xff,0x83,0xe0,0xff,0xff,0xff,0x80
,0x01,0xfc,0x07,0xc1,0xff,0xff,0xe3,0xc0
,0x01,0xe0,0x1f,0x01,0xff,0xff,0xc3,0xc0
,0x00,0x00,0xfe,0x01,0xff,0xff,0x87,0xe0
,0x00,0x03,0xf8,0x13,0xff,0xff,0x0f,0xe0
,0x00,0x1f,0xe0,0x73,0xff,0xfe,0x1f,0xe0
,0x00,0x7f,0x81,0xf3,0xff,0xfc,0x1f,0xe0
,0x00,0xfc,0x03,0xe3,0xef,0xf8,0x3f,0xe0
,0x00,0x60,0x0f,0xc3,0xc7,0xf0,0x7f,0xe0
,0x00,0x00,0x3f,0x03,0xc3,0xe0,0xff,0xe0
,0x00,0x00,0xfc,0x03,0xc1,0xc1,0xff,0xe0
,0x00,0x07,0xf0,0x13,0xe0,0x83,0xff,0xe0
,0x00,0x0f,0xc0,0x7b,0xf8,0x07,0xff,0xe0
,0x00,0x0f,0x01,0xf9,0xfc,0x0f,0xff,0xc0
,0x00,0x00,0x07,0xf1,0xfe,0x1f,0xff,0xc0

```
,0x00,0x00,0x1f,0xc0,0xff,0x3f,0xff,0x80
,0x00,0x00,0x7e,0x00,0xff,0xff,0xff,0x80
,0x00,0x00,0xfc,0x00,0x7f,0xff,0xff,0x00
,0x00,0x00,0xf0,0x1f,0x3f,0xff,0xfe,0x00
,0x00,0x00,0x00,0x7f,0x1f,0xff,0xfc,0x00
,0x00,0x00,0x01,0xff,0x8f,0xff,0xf8,0x00
,0x00,0x00,0x03,0xe0,0xe3,0xff,0xe0,0x00
,0x00,0x00,0x01,0x80,0x00,0x7f,0x00,0x00
};
//------------------------------------------------------------
#define FinPr_invalid_width 64
#define FinPr_invalid_height 64
const uint8_t PROGMEM FinPr_invalid_bits[] = {
  0x00,0x00,0x03,0xfe,0x00,0x00,0x00,0x00
,0x00,0x00,0x1f,0xff,0xe0,0x00,0x00,0x00
,0x00,0x00,0x7f,0xff,0xf8,0x00,0x00,0x00
,0x00,0x00,0xfc,0x00,0xfe,0x00,0x00,0x00
,0x00,0x03,0xe0,0x00,0x1f,0x00,0x00,0x00
,0x00,0x07,0xc0,0x00,0x07,0x80,0x00,0x00
,0x00,0x0f,0x80,0x00,0x03,0xe0,0x00,0x00
,0x00,0x0e,0x03,0xff,0x01,0xe0,0x00,0x00
,0x00,0x1c,0x1f,0xff,0xe0,0xf0,0x00,0x00
,0x00,0x3c,0x3f,0xff,0xf0,0x78,0x00,0x00
,0x00,0x78,0x7c,0x00,0xf8,0x3c,0x00,0x00
,0x00,0x70,0xf0,0x00,0x3c,0x1c,0x00,0x00
,0x00,0xe1,0xe0,0x00,0x1e,0x1c,0x00,0x00
,0x00,0xe1,0xc0,0x00,0x0f,0x0e,0x00,0x00
,0x00,0xc3,0x81,0xfc,0x07,0x0e,0x00,0x00
,0x00,0x03,0x83,0xff,0x07,0x8e,0x00,0x00
,0x00,0x07,0x07,0x8f,0x83,0x87,0x00,0x00
```

,0x00,0x0f,0x0f,0x03,0xc3,0x87,0x00,0x00
,0x00,0x1e,0x0e,0x01,0xc3,0x87,0x00,0x00
,0x00,0x3c,0x1c,0x00,0xe1,0x87,0x00,0x00
,0x00,0xf8,0x1c,0x30,0xe1,0x87,0x00,0x00
,0x07,0xf0,0x38,0x70,0xe1,0x86,0x00,0x00
,0x07,0xc0,0x78,0x70,0xe3,0x8e,0x00,0x00
,0x02,0x00,0xf0,0xf0,0xe3,0x8e,0x00,0x00
,0x00,0x01,0xe0,0xe0,0xe3,0x8e,0x00,0x00
,0x00,0x03,0xc1,0xe1,0xc3,0x8e,0x00,0x00
,0x00,0x0f,0x83,0xc3,0xc3,0x8e,0x00,0x00
,0x00,0x7f,0x07,0x83,0x83,0x0e,0x00,0x00
,0x07,0xfc,0x0f,0x07,0x83,0x0e,0x00,0x00
,0x07,0xf0,0x1e,0x0f,0x03,0x0e,0x00,0x00
,0x07,0x80,0x7c,0x1e,0x03,0x07,0x00,0x00
,0x00,0x00,0xf8,0x3c,0x03,0x87,0x80,0x00
,0x00,0x03,0xf0,0x78,0x03,0x83,0xc0,0x00
,0x00,0x1f,0xc0,0xf0,0x02,0x00,0x00,0x00
,0x00,0xff,0x01,0xe1,0xc0,0x00,0x00,0x00
,0x07,0xfc,0x03,0xc3,0xe1,0xff,0xc0,0x00
,0x07,0xe0,0x0f,0x87,0xc7,0xff,0xf0,0x00
,0x07,0x00,0x3f,0x0f,0x0f,0xff,0xf8,0x00
,0x00,0x00,0x7c,0x3e,0x1f,0xff,0xfe,0x00
,0x00,0x03,0xf8,0x7c,0x3f,0xff,0xff,0x00
,0x00,0x1f,0xe0,0xf0,0x7f,0xff,0xff,0x00
,0x00,0xff,0x83,0xe0,0xfe,0xff,0xbf,0x80
,0x01,0xfc,0x07,0xc0,0xfc,0x7f,0x1f,0xc0
,0x01,0xe0,0x1f,0x01,0xf8,0x3e,0x0f,0xc0
,0x00,0x00,0xfe,0x01,0xf8,0x1c,0x07,0xe0
,0x00,0x03,0xf8,0x13,0xf8,0x00,0x0f,0xe0
,0x00,0x1f,0xe0,0x73,0xfc,0x00,0x1f,0xe0

```
,0x00,0x7f,0x81,0xf3,0xfe,0x00,0x3f,0xe0
,0x00,0xfc,0x03,0xe3,0xff,0x00,0x7f,0xe0
,0x00,0x60,0x0f,0xc3,0xff,0x80,0xff,0xe0
,0x00,0x00,0x3f,0x03,0xff,0x00,0x7f,0xe0
,0x00,0x00,0xfc,0x03,0xfe,0x00,0x3f,0xe0
,0x00,0x07,0xf0,0x13,0xfc,0x00,0x1f,0xe0
,0x00,0x0f,0xc0,0x79,0xf8,0x08,0x0f,0xe0
,0x00,0x0f,0x01,0xf9,0xf8,0x1c,0x0f,0xc0
,0x00,0x00,0x07,0xf1,0xfc,0x3e,0x1f,0xc0
,0x00,0x00,0x1f,0xc0,0xfe,0x7f,0x3f,0x80
,0x00,0x00,0x7e,0x00,0xff,0xff,0xff,0x80
,0x00,0x00,0xfc,0x00,0x7f,0xff,0xff,0x00
,0x00,0x00,0xf0,0x1f,0x3f,0xff,0xfe,0x00
,0x00,0x00,0x00,0x7f,0x1f,0xff,0xfc,0x00
,0x00,0x00,0x01,0xff,0x8f,0xff,0xf8,0x00
,0x00,0x00,0x03,0xe0,0xe3,0xff,0xe0,0x00
,0x00,0x00,0x01,0x80,0x00,0x7f,0x00,0x00
};
//-----------------------------------------------------------
#define FinPr_failed_width 64
#define FinPr_failed_height 64
const uint8_t PROGMEM FinPr_failed_bits[] = {
0x00,0x00,0x3f,0xe0,0x00,0x00,0x00,0x00
,0x00,0x01,0xff,0xfe,0x00,0x00,0x00,0x00
,0x00,0x0f,0xc0,0x1f,0x80,0x00,0x00,0x00
,0x00,0x1e,0x00,0x03,0xc0,0x00,0x00,0x00
,0x00,0x78,0x00,0x00,0xf0,0x00,0x00,0x00
,0x00,0xe0,0x00,0x00,0x38,0x00,0x00,0x00
,0x01,0xc0,0x00,0x00,0x1c,0x00,0x00,0x00
,0x03,0x80,0x00,0x00,0x0e,0x00,0x00,0x00
```

,0x07,0x00,0x7f,0xe0,0x07,0x00,0x00,0x00
,0x06,0x01,0xff,0xf8,0x03,0x00,0x00,0x00
,0x0c,0x03,0xc0,0x3c,0x03,0x80,0x00,0x00
,0x1c,0x0f,0x00,0x0e,0x01,0x80,0x00,0x00
,0x18,0x0c,0x00,0x03,0x00,0xc0,0x00,0x00
,0x18,0x18,0x00,0x01,0x80,0xc0,0x00,0x00
,0x30,0x38,0x00,0x01,0xc0,0xe0,0x00,0x00
,0x30,0x30,0x0f,0x00,0xc0,0x60,0x00,0x00
,0x30,0x30,0x3f,0xc0,0xe0,0x60,0x00,0x00
,0x70,0x60,0x78,0xe0,0x60,0x60,0x00,0x00
,0x60,0x60,0x60,0x60,0x60,0x70,0x00,0x00
,0x60,0x60,0x60,0x60,0x60,0x30,0x00,0x00
,0x60,0x60,0x60,0x60,0x30,0x30,0x00,0x00
,0x60,0x60,0x60,0x30,0x30,0x20,0x00,0x00
,0x60,0x60,0x60,0x30,0x30,0x01,0xe0,0x00
,0x60,0x60,0x60,0x30,0x30,0x0f,0xfc,0x00
,0x60,0x60,0x60,0x30,0x30,0x3f,0xff,0x00
,0x60,0x60,0x60,0x30,0x18,0x78,0x03,0x80
,0x60,0x60,0x60,0x30,0x1c,0x60,0x01,0x80
,0x60,0x60,0x30,0x38,0x0c,0xc0,0x00,0xc0
,0x00,0x60,0x30,0x18,0x00,0xc0,0x00,0xc0
,0x00,0x60,0x30,0x18,0x00,0xc0,0x00,0xc0
,0x00,0xe0,0x30,0x0c,0x01,0xc0,0x00,0xe0
,0x00,0xc0,0x18,0x0e,0x01,0xc0,0x00,0xe0
,0x60,0xc0,0x18,0x07,0x01,0xc0,0x00,0xe0
,0x01,0xc0,0x1c,0x03,0x81,0xc0,0x00,0xe0
,0x01,0x80,0x0c,0x01,0xc1,0xc0,0x00,0xe0
,0x03,0x80,0x0e,0x00,0xf1,0xc0,0x00,0xe0
,0x0f,0x00,0x06,0x00,0x01,0xc0,0x00,0xe0
,0x3e,0x01,0x03,0x00,0x01,0xc0,0x00,0xe0

```
,0x30,0x03,0x83,0x80,0x1f,0xff,0xff,0xfe
,0x00,0x03,0x81,0xc0,0x3f,0xff,0xff,0xff
,0x00,0x07,0xc0,0xe0,0x30,0x00,0x00,0x03
,0x00,0x0e,0xc0,0x78,0x30,0x00,0x00,0x03
,0x00,0x3c,0x60,0x1e,0x30,0x00,0x00,0x03
,0x00,0x78,0x70,0x0f,0x30,0x00,0x00,0x03
,0x03,0xe0,0x38,0x03,0x30,0x00,0x00,0x03
,0x07,0x80,0x1c,0x00,0x30,0x00,0x00,0x03
,0xc0,0x00,0x0f,0x00,0x30,0x00,0x00,0x03
,0xc0,0x00,0x03,0x80,0x30,0x01,0xe0,0x03
,0x00,0x18,0x01,0xe0,0x30,0x03,0xf0,0x03
,0x00,0x18,0x00,0x7c,0x30,0x07,0x38,0x03
,0x00,0x0c,0x00,0x1f,0x30,0x06,0x18,0x03
,0x18,0x0e,0x00,0x07,0x30,0x06,0x18,0x03
,0x0c,0x07,0x80,0x00,0x30,0x07,0x38,0x03
,0x0e,0x03,0xc0,0x00,0x30,0x03,0x30,0x03
,0x07,0x00,0xf0,0x00,0x30,0x03,0x30,0x03
,0x03,0x00,0x7e,0x00,0x30,0x03,0x30,0x03
,0x01,0x80,0x1f,0xc0,0x30,0x03,0x30,0x03
,0x01,0xc0,0x03,0xe1,0x30,0x07,0xf8,0x03
,0x00,0xf0,0x00,0x01,0x30,0x03,0xf0,0x03
,0x00,0x38,0x00,0x00,0x30,0x00,0x00,0x03
,0x00,0x1e,0x00,0x00,0x30,0x00,0x00,0x03
,0x00,0x07,0xc0,0x00,0x30,0x00,0x00,0x03
,0x00,0x01,0xff,0x80,0x3f,0xff,0xff,0xff
,0x00,0x00,0x3f,0x80,0x1f,0xff,0xff,0xfe
};
//--------------------------------------------------------------
#define FinPr_scan_width 64
#define FinPr_scan_height 64
```

```
const uint8_t PROGMEM FinPr_scan_bits[] = {
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x1f,0xf8,0x00,0x00,0x00
,0x00,0x00,0x00,0x7f,0xff,0x00,0x00,0x00
,0x00,0x00,0x01,0xfc,0x7f,0xc0,0x00,0x00
,0x00,0x00,0x03,0xc0,0x03,0xe0,0x00,0x00
,0x00,0x00,0x07,0x80,0x00,0xf0,0x00,0x00
,0x00,0x00,0x0e,0x00,0x00,0x3c,0x00,0x00
,0x00,0x00,0x1c,0x1f,0xfc,0x1c,0x00,0x00
,0x00,0x00,0x38,0x7f,0xfe,0x0e,0x00,0x00
,0x00,0x00,0x78,0xf8,0x0f,0x87,0x00,0x00
,0x00,0x00,0x71,0xe0,0x03,0xc7,0x00,0x00
,0x00,0x00,0xe3,0x80,0x01,0xc3,0x80,0x00
,0x00,0x00,0xc3,0x83,0xc0,0xe3,0x80,0x00
,0x00,0x00,0xc7,0x0f,0xf0,0x71,0x80,0x00
,0x00,0x00,0x06,0x1f,0xf8,0x71,0xc0,0x00
,0x00,0x00,0x0e,0x1c,0x3c,0x31,0xc0,0x00
,0x00,0x00,0x1c,0x38,0x1c,0x31,0xc0,0x00
,0x00,0x00,0x38,0x70,0x0e,0x39,0xc0,0x00
,0x00,0x01,0xf0,0x71,0x8e,0x39,0xc0,0x00
,0x00,0x03,0xe0,0xe1,0x86,0x31,0xc0,0x00
,0x00,0x03,0x81,0xe3,0x8e,0x31,0x80,0x00
,0x00,0x00,0x03,0xc3,0x8e,0x33,0x80,0x00
,0x00,0x00,0x07,0x87,0x0c,0x73,0x80,0x00
,0x00,0x00,0x1f,0x0e,0x1c,0x73,0x80,0x00
,0x7f,0xff,0xff,0xff,0xff,0xff,0xff,0xfe
```

,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff
,0x7f,0xff,0xff,0xff,0xff,0xff,0xff,0xfe
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x03,0xf0,0x1e,0x3e,0x1c,0x00,0x00
,0x00,0x03,0x80,0x7c,0x77,0x0f,0x00,0x00
,0x00,0x00,0x01,0xf0,0xe3,0x07,0xc0,0x00
,0x00,0x00,0x07,0xe3,0xc3,0x81,0xf0,0x00
,0x00,0x00,0x3f,0x87,0x81,0xc0,0x60,0x00
,0x00,0x01,0xfc,0x1f,0x00,0xf0,0x00,0x00
,0x00,0x01,0xe0,0x3c,0x00,0x7c,0x00,0x00
,0x00,0x00,0x00,0xf8,0x78,0x1f,0x00,0x00
,0x00,0x00,0x07,0xe0,0xfc,0x0f,0xc0,0x00
,0x00,0x00,0x3f,0x83,0xef,0x03,0xc0,0x00
,0x00,0x00,0xfc,0x0f,0x87,0x80,0x00,0x00
,0x00,0x00,0x70,0x1f,0x03,0xe0,0x00,0x00
,0x00,0x00,0x00,0x7c,0x00,0xf8,0x00,0x00
,0x00,0x00,0x01,0xf0,0x00,0x3e,0x00,0x00
,0x00,0x00,0x0f,0xc0,0xf8,0x0f,0x00,0x00
,0x00,0x00,0x1f,0x03,0xfe,0x02,0x00,0x00
,0x00,0x00,0x0c,0x0f,0x8f,0x80,0x00,0x00
,0x00,0x00,0x00,0x3f,0x03,0xe0,0x00,0x00
,0x00,0x00,0x00,0xf8,0x00,0xf0,0x00,0x00
,0x00,0x00,0x01,0xe0,0x00,0x30,0x00,0x00
,0x00,0x00,0x01,0xc0,0xf8,0x00,0x00,0x00
,0x00,0x00,0x00,0x07,0xfe,0x00,0x00,0x00
,0x00,0x00,0x00,0x0f,0x8e,0x00,0x00,0x00
,0x00,0x00,0x00,0x06,0x00,0x00,0x00,0x00

```
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
};

void setup(){
  Serial.begin(115200);
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
  display.display();
  delay(2000); // Pause for 2 seconds
  display.clearDisplay();
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED){
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();
  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;
  if (Firebase.signUp(&config, &auth, "", "")){
    Serial.println("ok");
```

```
  signupOK = true;
 }
 else{
  Serial.printf("%s\n", config.signer.signupError.message.c_str());
 }
 config.token_status_callback = tokenStatusCallback; //see addons/TokenHelper.h
 Firebase.begin(&config, &auth);
 Firebase.reconnectWiFi(true);
 finger.begin(57600);
 Serial.println("\n\nAdafruit finger detect test");

 if (finger.verifyPassword()) {
  Serial.println("Found fingerprint sensor!");
  display.clearDisplay();
  display.drawBitmap( 34, 0, FinPr_valid_bits, FinPr_valid_width, FinPr_valid_height,
WHITE);
  display.display();
 } else {
  Serial.println("Did not find fingerprint sensor :(");
  display.clearDisplay();
  display.drawBitmap( 32, 0, FinPr_failed_bits, FinPr_failed_width, FinPr_failed_height,
WHITE);
  display.display();
  while (1) { delay(1); }
 }

 finger.getTemplateCount();
 Serial.print("Sensor contains "); Serial.print(finger.templateCount); Serial.println("
templates");
 Serial.println("Waiting for valid finger...");
```

```
  pinMode(D4, OUTPUT);
  digitalWrite(D4,LOW);
}

void loop(){
  ChecktoAddID();
  delay(1000);
}

int getFingerprintID() {
  uint8_t p = finger.getImage();
  switch (p) {
    case FINGERPRINT_OK:
      //Serial.println("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      //Serial.println("No finger detected");
      return 0;
    case FINGERPRINT_PACKETRECIEVEERR:
      //Serial.println("Communication error");
      return -2;
    case FINGERPRINT_IMAGEFAIL:
      //Serial.println("Imaging error");
      return -2;
    default:
      //Serial.println("Unknown error");
      return -2;
  }
  // OK success!
```

```
p = finger.image2Tz();
switch (p) {
  case FINGERPRINT_OK:
    //Serial.println("Image converted");
    break;
  case FINGERPRINT_IMAGEMESS:
    //Serial.println("Image too messy");
    return -1;
  case FINGERPRINT_PACKETRECIEVEERR:
    //Serial.println("Communication error");
    return -2;
  case FINGERPRINT_FEATUREFAIL:
    //Serial.println("Could not find fingerprint features");
    return -2;
  case FINGERPRINT_INVALIDIMAGE:
    //Serial.println("Could not find fingerprint features");
    return -2;
  default:
    //Serial.println("Unknown error");
    return -2;
}
// OK converted!
p = finger.fingerFastSearch();
if (p == FINGERPRINT_OK) {
  //Serial.println("Found a print match!");
} else if (p == FINGERPRINT_PACKETRECIEVEERR) {
  //Serial.println("Communication error");
  return -2;
} else if (p == FINGERPRINT_NOTFOUND) {
  //Serial.println("Did not find a match");
```

```
    return -1;
  } else {
    //Serial.println("Unknown error");
    return -2;
  }
  // found a match!
  //Serial.print("Found ID #"); Serial.print(finger.fingerID);
  //Serial.print(" with confidence of "); Serial.println(finger.confidence);

  return finger.fingerID;
}

void DisplayFingerprintID(){
  //Fingerprint has been detected
  if (FingerID > 0){
    display.clearDisplay();
    display.drawBitmap( 34, 0, FinPr_valid_bits, FinPr_valid_width, FinPr_valid_height,
WHITE);
    display.display();
    if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis > 1000 ||
sendDataPrevMillis == 0)){
    sendDataPrevMillis = millis();
    if (Firebase.RTDB.setInt(&fbdo, "mainbucket/command",FingerID)){
      Serial.println("PATH: " + fbdo.dataPath());
      Serial.println("TYPE: " + fbdo.dataType());
    }
    else {
      Serial.println("Failed REASON: " + fbdo.errorReason());
    }
  }
```

```
  }
  //----------------------------------------
  //No finger detected
  else if (FingerID == 0){
    display.clearDisplay();
    display.drawBitmap( 32, 0, FinPr_start_bits, FinPr_start_width, FinPr_start_height,
WHITE);
    display.display();
  }
  //----------------------------------------
  //Didn't find a match
  else if (FingerID == -1){
    display.clearDisplay();
    display.drawBitmap( 34, 0, FinPr_invalid_bits, FinPr_invalid_width, FinPr_invalid_height,
WHITE);
    display.display();
  }
  //----------------------------------------
  //Didn't find the scanner or there an error
  else if (FingerID == -2){
    display.clearDisplay();
    display.drawBitmap( 32, 0, FinPr_failed_bits, FinPr_failed_width, FinPr_failed_height,
WHITE);
    display.display();
  }
}

void ChecktoAddID(){
  if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis > 1000 ||
sendDataPrevMillis == 0)){
```

```
sendDataPrevMillis = millis();
if (Firebase.RTDB.getString(&fbdo, "/account/command/command"))
{
  intValue = fbdo.stringData();
  String mySubString = intValue.substring(0, 3);
  String SubString = intValue.substring(3, 6);
  id = SubString.toInt();
  Serial.println(intValue);
  Serial.println(mySubString);
  if (mySubString == "add")
  {
    getFingerprintEnroll();
    delay(1000);
  }
  else if (mySubString == "ver")
  {
    FingerID = getFingerprintID();
    delay(50);
    DisplayFingerprintID();
    delay(1000);
  }
  else if (mySubString == "del")
  {
    deleteFingerprint(id);
    delay(1000);
  }
  else if (mySubString == "ope")
  {
    digitalWrite(D4, HIGH);
    delay(1000);
```

```
    }
    else if (mySubString == "clo")
    {
      digitalWrite(D4, LOW);
      delay(1000);
    }
   delay(100);
   }
   else {
    Serial.println(fbdo.errorReason());
   }
   delay(100);
 }
}

uint8_t getFingerprintEnroll() {

 int p = -1;
 display.clearDisplay();
 display.drawBitmap( 34, 0, FinPr_scan_bits, FinPr_scan_width, FinPr_scan_height,
WHITE);
 display.display();
 while (p != FINGERPRINT_OK) {
  p = finger.getImage();
  switch (p) {
  case FINGERPRINT_OK:
   //Serial.println("Image taken");
   display.clearDisplay();
   display.drawBitmap( 34, 0, FinPr_valid_bits, FinPr_valid_width, FinPr_valid_height,
WHITE);
```

```
      display.display();
      break;
    case FINGERPRINT_NOFINGER:
      //Serial.println(".");
      display.setTextSize(1);             // Normal 2:2 pixel scale
      display.setTextColor(WHITE);        // Draw white text
      display.setCursor(0,0);             // Start at top-left corner
      display.print(F("scanning"));
      display.display();
      break;
    case FINGERPRINT_PACKETRECIEVEERR:
      display.clearDisplay();
      display.drawBitmap(         34,         0,         FinPr_invalid_bits,         FinPr_invalid_width,
FinPr_invalid_height, WHITE);
      display.display();
      break;
    case FINGERPRINT_IMAGEFAIL:
      Serial.println("Imaging error");
      break;
    default:
      Serial.println("Unknown error");
      break;
    }
  }

  // OK success!

  p = finger.image2Tz(1);
  switch (p) {
    case FINGERPRINT_OK:
```

```
      display.clearDisplay();
      display.drawBitmap( 34, 0, FinPr_valid_bits, FinPr_valid_width, FinPr_valid_height,
WHITE);
      display.display();
      break;
    case FINGERPRINT_IMAGEMESS:
      display.clearDisplay();
      display.drawBitmap( 34, 0, FinPr_invalid_bits, FinPr_invalid_width,
FinPr_invalid_height, WHITE);
      display.display();
      return p;
    case FINGERPRINT_PACKETRECIEVEERR:
      Serial.println("Communication error");
      return p;
    case FINGERPRINT_FEATUREFAIL:
      Serial.println("Could not find fingerprint features");
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      Serial.println("Could not find fingerprint features");
      return p;
    default:
      Serial.println("Unknown error");
      return p;
  }
  display.clearDisplay();
  display.setTextSize(2);             // Normal 2:2 pixel scale
  display.setTextColor(WHITE);        // Draw white text
  display.setCursor(0,0);             // Start at top-left corner
  display.print(F("Remove"));
  display.setCursor(0,20);
```

```
  display.print(F("finger"));
  display.display();
  //Serial.println("Remove finger");
  delay(2000);
  p = 0;
  while (p != FINGERPRINT_NOFINGER) {
    p = finger.getImage();
  }
  Serial.print("ID "); Serial.println(id);
  p = -1;
  display.clearDisplay();
  display.drawBitmap(  34,  0,  FinPr_scan_bits,  FinPr_scan_width,  FinPr_scan_height,
WHITE);
  display.display();
  while (p != FINGERPRINT_OK) {
    p = finger.getImage();
    switch (p) {
    case FINGERPRINT_OK:
      //Serial.println("Image taken");
      display.clearDisplay();
      display.drawBitmap(  34,  0,  FinPr_valid_bits,  FinPr_valid_width,  FinPr_valid_height,
WHITE);
      display.display();
      break;
    case FINGERPRINT_NOFINGER:
      //Serial.println(".");
      display.setTextSize(1);            // Normal 2:2 pixel scale
      display.setTextColor(WHITE);       // Draw white text
      display.setCursor(0,0);            // Start at top-left corner
      display.print(F("scanning"));
```

```
    display.display();
    break;
   case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
    break;
   case FINGERPRINT_IMAGEFAIL:
    Serial.println("Imaging error");
    break;
   default:
    Serial.println("Unknown error");
    break;
   }
  }

  // OK success!

  p = finger.image2Tz(2);
  switch (p) {
   case FINGERPRINT_OK:
    //Serial.println("Image converted");
    display.clearDisplay();
    display.drawBitmap( 34, 0, FinPr_valid_bits, FinPr_valid_width, FinPr_valid_height,
WHITE);
    display.display();
    break;
   case FINGERPRINT_IMAGEMESS:
    Serial.println("Image too messy");
    return p;
   case FINGERPRINT_PACKETRECIEVEERR:
    Serial.println("Communication error");
```

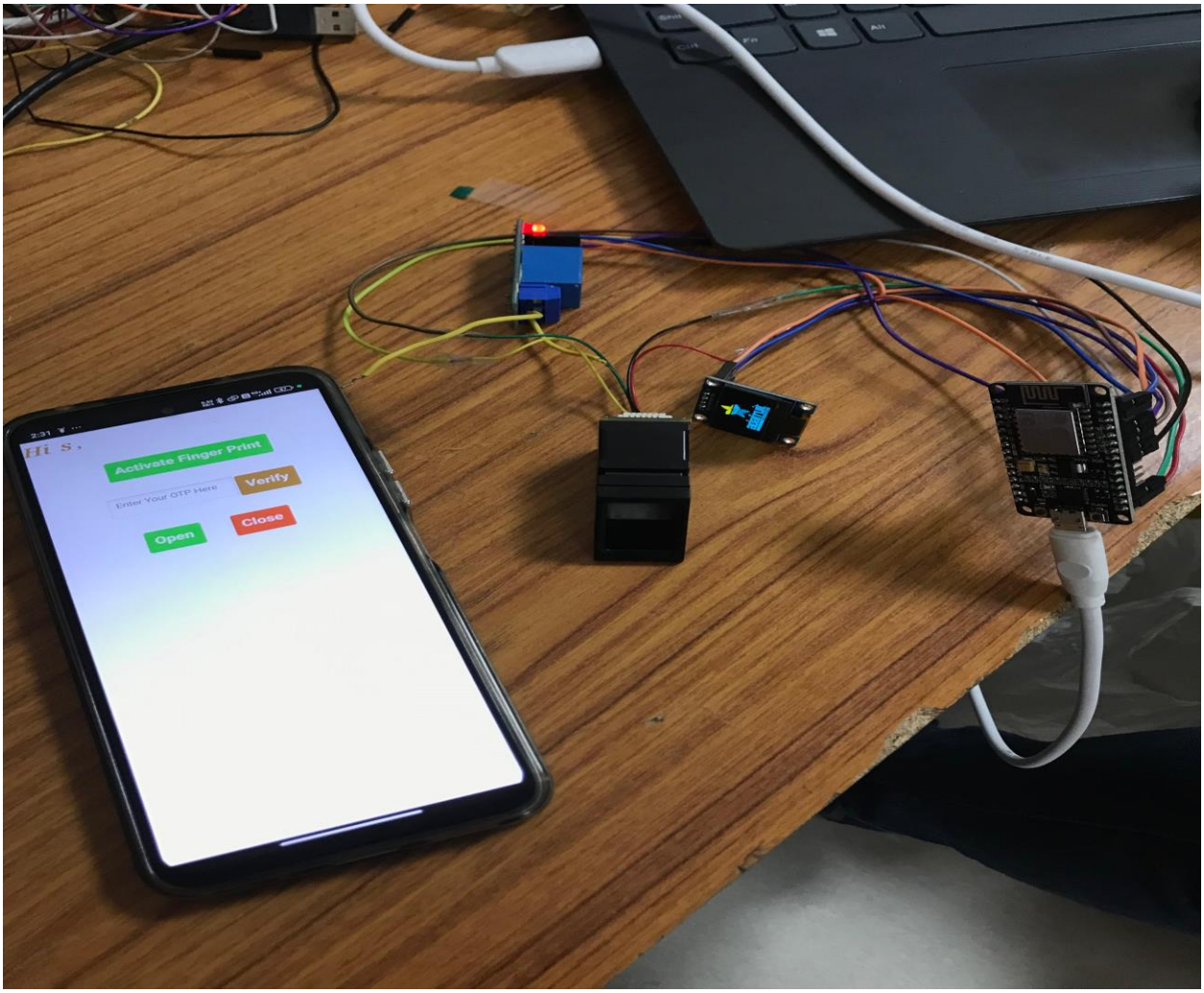
```
    return p;
   case FINGERPRINT_FEATUREFAIL:
    Serial.println("Could not find fingerprint features");
    return p;
   case FINGERPRINT_INVALIDIMAGE:
    Serial.println("Could not find fingerprint features");
    return p;
   default:
    Serial.println("Unknown error");
    return p;
  }

  // OK converted!
  Serial.print("Creating model for #");  Serial.println(id);

  p = finger.createModel();
  if (p == FINGERPRINT_OK) {
   //Serial.println("Prints matched!");
   display.clearDisplay();
   display.drawBitmap(  34,  0,  FinPr_valid_bits,  FinPr_valid_width,  FinPr_valid_height,
WHITE);
   display.display();
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
   Serial.println("Communication error");
   return p;
  } else if (p == FINGERPRINT_ENROLLMISMATCH) {
   Serial.println("Fingerprints did not match");
   return p;
  } else {
   Serial.println("Unknown error");
```

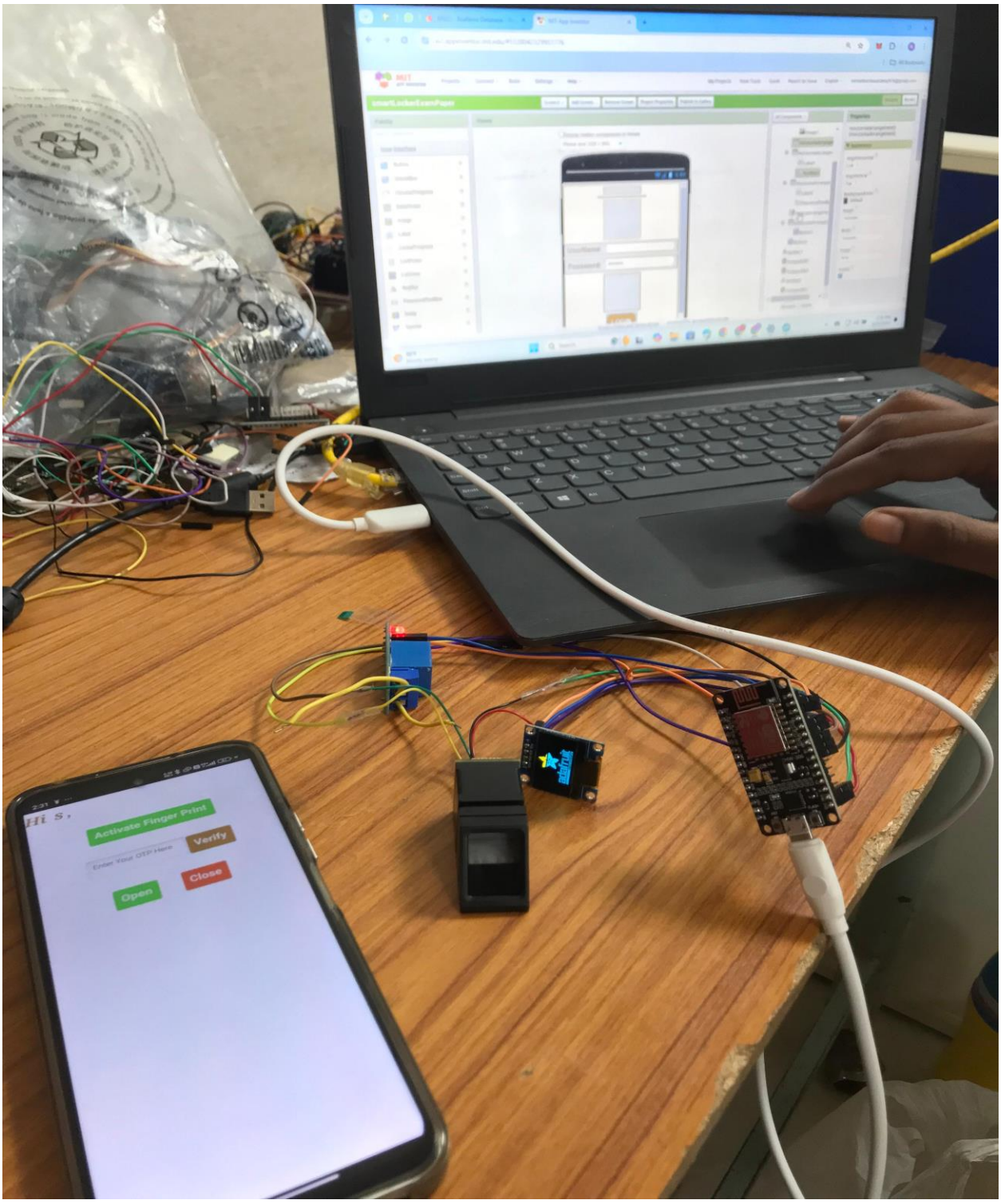
```
    return p;
  }

  Serial.print("ID "); Serial.println(id);
  p = finger.storeModel(id);
  if (p == FINGERPRINT_OK)
  {
    //Serial.println("Stored!");
    display.clearDisplay();
    display.drawBitmap( 34, 0, FinPr_valid_bits, FinPr_valid_width, FinPr_valid_height,
WHITE);
    display.display();
    if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis > 1000 ||
sendDataPrevMillis == 0)){
    sendDataPrevMillis = millis();
    if (Firebase.RTDB.setString(&fbdo, "account/command/command","done")){
      Serial.println("PATH: " + fbdo.dataPath());
      Serial.println("TYPE: " + fbdo.dataType());
    }
    else {
      Serial.println("Failed REASON: " + fbdo.errorReason());
    }
    }
    return p;
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    Serial.println("Communication error");
    return p;
  } else if (p == FINGERPRINT_BADLOCATION) {
    Serial.println("Could not store in that location");
    return p;
```

```
  } else if (p == FINGERPRINT_FLASHERR) {
    Serial.println("Error writing to flash");
    return p;
  } else {
    Serial.println("Unknown error");
    return p;
  }
}
uint8_t deleteFingerprint( int id) {
  uint8_t p = -1;

  p = finger.deleteModel(id);

  if (p == FINGERPRINT_OK) {
    //Serial.println("Deleted!");
    display.clearDisplay();
    display.setTextSize(2);            // Normal 2:2 pixel scale
    display.setTextColor(WHITE);       // Draw white text
    display.setCursor(0,0);            // Start at top-left corner
    display.print(F("Deleted!\n"));
    display.display();
    return p;
  } else if (p == FINGERPRINT_PACKETRECIEVEERR) {
    //Serial.println("Communication error");
    display.clearDisplay();
    display.setTextSize(1);            // Normal 1:1 pixel scale
    display.setTextColor(WHITE);       // Draw white text
    display.setCursor(0,0);            // Start at top-left corner
    display.print(F("Communication error!\n"));
    display.display();
```

```
    return p;
  } else if (p == FINGERPRINT_BADLOCATION) {
    //Serial.println("Could not delete in that location");
    display.clearDisplay();
    display.setTextSize(1);            // Normal 1:1 pixel scale
    display.setTextColor(WHITE);       // Draw white text
    display.setCursor(0,0);            // Start at top-left corner
    display.print(F("Could not delete in that location!\n"));
    display.display();
    return p;
  } else if (p == FINGERPRINT_FLASHERR) {
    //Serial.println("Error writing to flash");
    display.clearDisplay();
    display.setTextSize(1);            // Normal 1:1 pixel scale
    display.setTextColor(WHITE);       // Draw white text
    display.setCursor(0,0);            // Start at top-left corner
    display.print(F("Error writing to flash!\n"));
    display.display();
    return p;
  } else {
    //Serial.print("Unknown error: 0x"); Serial.println(p, HEX);
    display.clearDisplay();
    display.setTextSize(2);            // Normal 2:2 pixel scale
    display.setTextColor(WHITE);       // Draw white text
    display.setCursor(0,0);            // Start at top-left corner
    display.print(F("Unknown error:\n"));
    display.display();
    return p;
    }
}
```

Result:


Activate Finger Print
Verify
Enter Your OTP Here
Open
Close


Activate Finger Print
Verify
Close
Open