TABLE OF CONTENTS

        - Other important functions (menu navigation, vote casting, results checking, etc.)
    - Instructions for customizing the code (if necessary)
- USER GUIDE
    - Step-by-step instructions for using the voting system
        - Powering on and off
        - Navigating the menu
        - Enrolling voters (fingerprint or RFID)
        - Casting votes
        - Checking results
    - Troubleshooting common issues and FAQs
- CONCLUSION
    - Summary of the voting system's features and benefits
    - Future improvements and enhancements
- REFERENCES
    - Arduino documentation and tutorials.
    - Library documentation for the specific libraries used in the project.
    - Datasheets for the hardware components (e.g., RFID reader, fingerprint sensor, etc.).
    - External articles, blog posts, or research papers related to the voting system, its components, or similar projects.
- APPENDICES

LIST OF FIGURES

INTRODUCTION

The Arduino-based voting system is an electronic voting platform designed to provide a secure, efficient, and user-friendly way to cast votes in various types of elections or polls. The system utilizes an Arduino microcontroller, along with various input devices, such as a rotary encoder for menu navigation, and RFID and fingerprint sensors for voter identification and authentication. The results are displayed on an LCD screen, offering real-time access to voting outcomes.

Key features of the Arduino-based voting system include:

- Secure voter identification using RFID cards and fingerprint sensors
- Intuitive menu navigation using a rotary encoder
- Real-time display of voting results on an LCD screen
- Easy integration with various Arduino boards and input/output devices

The main benefits of using the Arduino-based voting system are:

- Improved security and reliability compared to traditional paper-based voting methods
- Faster and more efficient vote casting and tallying processes
- Cost-effective solution for small-scale elections or polls
- Customizable and expandable design, allowing for future enhancements and upgrades

The Arduino-based voting system is suitable for a wide range of users, including educational institutions, local communities, and organizations looking to conduct elections or polls in a secure and efficient manner. With its user-friendly interface and secure identification methods, the system can be easily adapted to various voting scenarios and requirements.

SYSTEM OVERVIEW

The Arduino-based voting system is an innovative electronic voting solution that aims to facilitate secure and efficient voting processes while ensuring accurate recording and displaying of results. This section provides an in-depth description of the system's hardware and software components, as well as their interconnections and interactions, in a manner suitable for patent documentation.

System Architecture

The System Architecture of the Arduino-based voting system illustrates the organization and interconnection of the various hardware and software components. It includes the following main elements:

- Arduino board as the central processing unit
- Input devices for user interaction and voter authentication
- Display for presenting information to users
- Real-time clock for maintaining accurate timestamps
- Power supply for providing the necessary voltage and current to the system components

Hardware Components

a. Arduino Board

The Arduino board (specify the model) serves as the central processing unit of the voting system, executing the software code and managing the communication between the various input devices, display, and real-time clock. The board is selected based on its processing power, memory capacity, input/output capabilities, and compatibility with the other system components.

b. Input Devices

The Arduino-based voting system employs a variety of input devices for user interaction and voter authentication:

- Rotary Encoder: Allows users to navigate through the menu options and make selections. It provides both rotational and push-button inputs, enabling intuitive and efficient user interactions.
- RFID Reader: Utilizes radio-frequency identification technology to read unique voter identification cards, ensuring secure voter authentication and preventing unauthorized access to the voting system.
- Fingerprint Sensor: Offers an additional layer of security by verifying the voter's fingerprint, further reducing the risk of voter fraud and unauthorized access.

c. Display

An LCD (Liquid Crystal Display) is used to present information to users, such as menu options, voting instructions, and result updates. The display is chosen based on its readability, size, resolution, and compatibility with the Arduino board and LiquidCrystal_I2C library.

d. Other Components

Real-Time Clock (RTC): Ensures accurate timestamps are maintained throughout the voting process, facilitating precise record-keeping and result reporting.

Power Supply: Provides the necessary voltage and current to the system components, ensuring stable and reliable operation. The power supply is selected based on its capacity, efficiency, and compatibility with the system's voltage and current requirements.

Software Components

The Arduino-based voting system relies on a combination of pre-existing libraries and custom-developed code to implement its various functionalities:

a. Arduino Libraries

Several standard Arduino libraries are used to facilitate the communication and control of the system's hardware components:

- Wire: Enables communication between the Arduino board and the RTC module using the I2C protocol.
- LiquidCrystal_I2C: Facilitates the control and display of text and graphics on the LCD.

Other libraries specific to the input devices, such as the RFID reader and fingerprint sensor, may also be employed.

b. Key Algorithms and Functionalities

The custom-developed software code for the Arduino-based voting system implements the following key algorithms and functionalities:

- Menu Navigation: Manages the user's interactions with the menu options and sub-menus, enabling the selection of various voting-related tasks, such as enrolling voters, casting votes, and checking results.
- Vote Casting: Handles the process of recording and storing individual votes securely, ensuring that each voter can only cast one vote and that the vote data is accurately maintained.
- Results Checking: Retrieves and displays the voting results, allowing users to monitor the outcome of the election in real-time.
- oter Authentication: Integrates the RFID and fingerprint-based voter authentication methods, ensuring that only authorized voters can access and use the voting system.

HARDWARE SETUP

This section provides a comprehensive description of each hardware component, their connections to the Arduino board, and essential safety considerations and best practices for the Arduino-based voting system.

Detailed Description of Each Hardware Component

a. **Arduino Board**

The Arduino board (specify the model) is the central processing unit of the voting system. It is responsible for executing the software code, managing communication between input devices, display, and real-time clock. The board's features include a microcontroller, digital input/output pins, analog input pins, power supply pins, and communication interfaces, such as I2C, SPI, and UART.

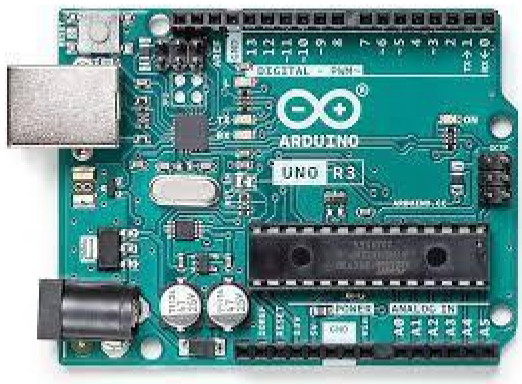

FIG 1: ARDUINO BOARD

b. Rotary Encoder

The rotary encoder is a user input device that enables voters to navigate through the menu options and make selections. It consists of a rotating knob that generates pulses when turned, and a push-button switch for confirming the selections. The encoder is connected to the Arduino board's digital input pins and requires minimal external components for its operation.

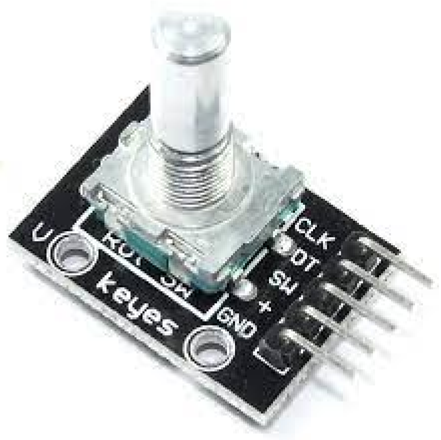

FIG 2: ROTARTY ENCODER

c. RFID Reader

The RFID reader is an essential component for secure voter authentication. It uses radio-frequency identification technology to read the unique identification data stored on the voter's RFID card. The reader is typically connected to the Arduino board using an SPI or UART interface, depending on the specific model.

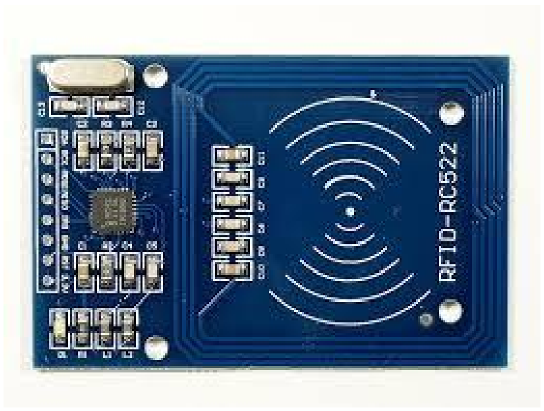

FIG 3: RFID MODULE

d. Fingerprint Sensor

The fingerprint sensor provides an additional layer of security for the voting system by verifying the voter's fingerprint. It typically includes an optical or capacitive sensing element, a processing unit, and a communication interface (UART or SPI). The sensor is connected to the Arduino board through the appropriate interface and requires power supply connections.

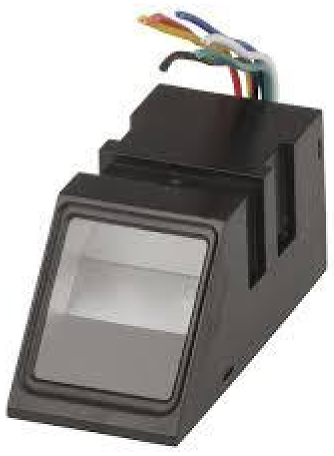

FIG 4: FINGERPRINT SENSOR

e. LCD Display

The LCD (Liquid Crystal Display) presents information to the user, such as menu options, voting instructions, and result updates. It typically requires connections to the Arduino board's digital pins or I2C interface, as well as power supply connections

FIG 5: LCD 1602 WITH I2C FIG 6: REAL TIME CLOCK MODULE FIG 7: OVERALL CIRCUIT DIAGRAM

f. Real-Time Clock (RTC)

The RTC module ensures accurate timestamps throughout the voting process. It typically features an I2C interface for communication with the Arduino board and a backup battery to maintain the time even when the main power is disconnected.

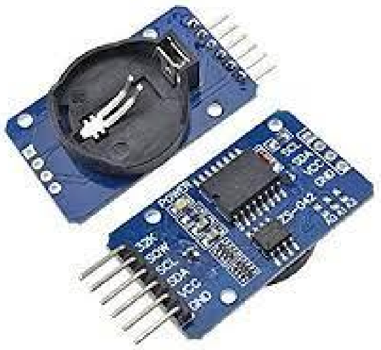

FIG 6: REAL TIME CLOCK MODULE

g. Power Supply

The power supply provides the necessary voltage and current for the system components. It should be chosen based on the system's overall power requirements and compatible with the voltage and current specifications of each component.

Connection Diagram

A detailed connection diagram provided, illustrating how each hardware component is connected to the Arduino board. The diagram clearly shows the connections between the Arduino's input/output pins, communication interfaces, and power supply pins and the respective pins or terminals on each hardware component.

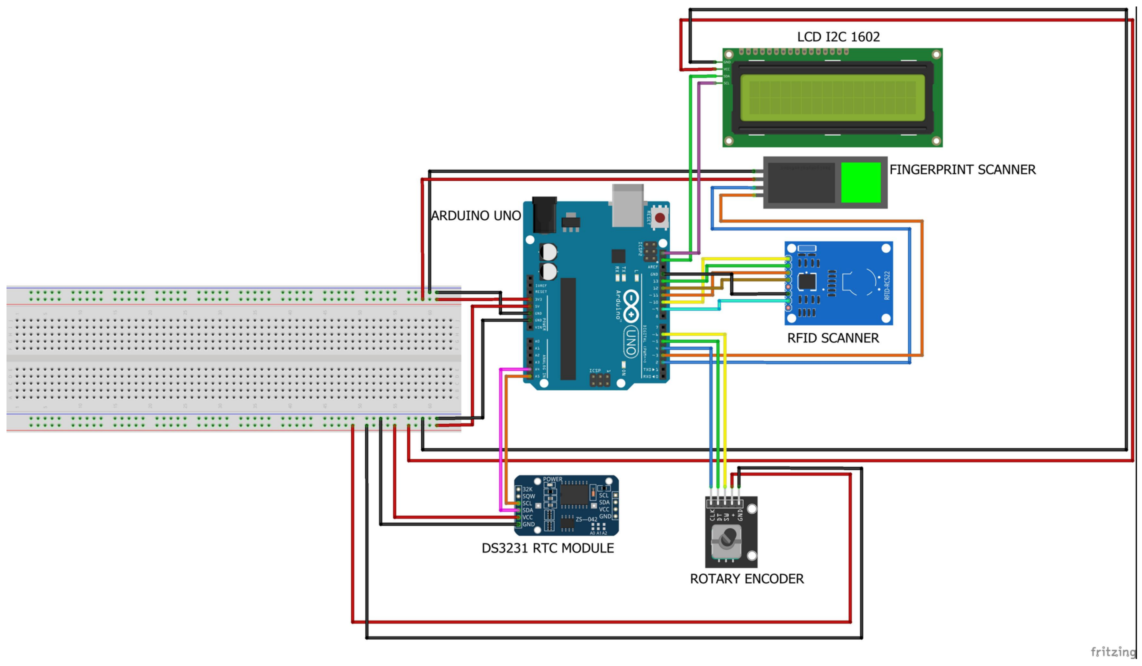

FIG 7: OVERALL CIRCUIT DIAGRAM

Safety Considerations and Best Practices

When setting up the hardware for the Arduino-based voting system, the following safety considerations and best practices were observed:

a. Ensure proper power supply selection and connections to avoid overloading the system or causing damage to the components.

b. Use appropriate connectors and wiring to maintain reliable and secure connections between the components.

c. Verify the correct polarity and voltage levels for each component before powering up the system.

d. Implement proper grounding and shielding techniques to minimize electromagnetic interference and ensure reliable operation of the system.

e. Keep the system components and connections organized and protected from physical damage, dust, and moisture.

f. Follow the manufacturer's recommendations and guidelines for each hardware component to ensure their correct and safe operation.

SOFTWARE SETUP

This section provides an overview of the software setup for the Arduino-based voting system, including the required libraries, code structure, key functions, and instructions for customization.

Required Arduino Libraries and Installation Instructions

To properly run the voting system code, several Arduino libraries are necessary. The following libraries should be downloaded and installed through the Arduino IDE's Library Manager or manually by adding them to the "libraries" folder in the Arduino IDE installation directory:

1. **Wire**: A library for I2C communication, used for connecting the LCD display and RTC module.
2. LiquidCrystal_I2C: A library for controlling the I2C-based LCD display.
3. MFRC522: A library for interfacing with the RFID reader module.
4. Adafruit_Fingerprint: A library for interfacing with the fingerprint sensor module.
5. (Optional) RTClib: A library for managing the real-time clock module.

Brief Explanation of the Main Code Structure

The voting system code is structured into several key sections:

- **Global variables and constants**: These define the system's settings, hardware connections, and other necessary values.
- **Function declarations**: Function prototypes for the various functions used in the code, such as menu navigation, vote casting, and results checking.

- **Setup function:** Initializes the hardware components, libraries, and sets up the initial state of the system.
- **Loop function:** The main program loop that continuously checks for user input and updates the system accordingly.
- **Other important functions:** Functions that implement specific tasks, such as enrolling a fingerprint, casting a vote, or checking the results.

Setup Function

The setup function initializes the hardware components and libraries used in the voting system. It configures the input/output pins, initializes the communication interfaces, and sets up the LCD display, fingerprint sensor, and RFID reader.

Loop Function

The loop function is the main program loop that runs continuously during the operation of the voting system. It monitors user input from the rotary encoder and button, updates the menu display, and triggers the appropriate functions based on the user's selections.

Other Important Functions

The voting system code includes several key functions that implement specific tasks:

Menu navigation: Functions that handle the navigation through the main menu and sub-menus, updating the LCD display with the current menu option.

Vote casting: A function that processes the voting process, verifying the voter's identity and allowing them to cast their vote.

Results checking: A function that allows authorized users to check the results of the voting process.

Enrolling a fingerprint: A function for enrolling new fingerprints into the system, typically performed by an administrator.

Instructions for Customizing the Code (if necessary)

If you need to customize the voting system's code to suit specific requirements, follow these guidelines:

- Modify the global variables and constants to match your hardware setup or system preferences.
- Add or modify functions to implement additional features or modify existing ones.
- Update the loop function and other relevant functions to reflect any changes you made to the code structure.
- Test your changes thoroughly to ensure the system operates correctly and securely.
- Remember to document any changes you make and provide comments within the code to explain your modifications for future reference or maintenance.

USER GUIDE

This user guide provides step-by-step instructions for operating the Arduino-based voting system, covering key tasks such as powering the system on and off, navigating the menu, enrolling voters, casting votes, checking results, and troubleshooting common issues.

Powering On and Off

To power on the voting system, connect the power supply to the Arduino board, ensuring that all hardware components are connected securely. The system should initialize, and the LCD display will show the main menu. To power off the system, simply disconnect the power supply.

Navigating the Menu

The main menu is navigated using the rotary encoder and button. Turning the rotary encoder allows you to cycle through the available options, which will be displayed on the LCD screen. When directly at an option, you should see '>' sign signifying that you are about to select that option ,To select an option, press the button on the rotary encoder.

Enrolling Voters (Fingerprint)

To enroll a new voter, follow these steps:

a. Navigate to the "Enroll" option in the main menu and press the button to enter the submenu. b. Choose "Enroll FP" to enroll the voter

c. Follow the on-screen instructions for the enrollment process, which may involve scanning the voter's fingerprint.

d. Once the enrollment is successful, the system will return to the main menu.

Casting Votes

Voters can cast their votes using the following steps:

a. Navigate to the "Vote" option in the main menu and press the button to enter the submenu. b. Verify the voter's identity using the fingerprint enrolled method by following the on-screen instructions. c. Once the voter's identity is verified, they can cast their vote using the rotary encoder to select their preferred candidate or option.

d. Press the button to confirm the vote. The system will record the vote and return to the main menu.

Checking Results

To check the voting results, authorized users can follow these steps:

a. Navigate to the "Results" option in the main menu and press the button to enter the submenu.

b. Verify the authorized user's identity using the RFID method by following the on-screen instructions.

c. Once access is granted, the system will display the voting results on the LCD screen.

d. Wait for 5 seconds to return to the main menu.

Troubleshooting Common Issues and FAQs

Q: The system is not responding or appears to be frozen.

A: Check the power supply and connections between components. If everything appears to be connected correctly, try resetting the system by disconnecting and reconnecting the power supply.

Q: The LCD display is not showing anything or is displaying garbled text.

A: Ensure that the LCD is connected correctly to the Arduino board and that the I2C address is properly set in the code.

Q: The RFID reader or fingerprint sensor is not detecting input.

A: Check the connections between the sensor and the Arduino board. Make sure the sensor is properly powered and initialized in the code.

For any other issues or questions, consult the system documentation, hardware and software setup guides.

CONCLUSION

In summary, the Arduino-based voting system provides a secure, reliable, and user-friendly solution for small-scale elections or polls. Key features of the system include multiple voter authentication methods (fingerprint and RFID), an intuitive menu-driven interface, and efficient vote storage and retrieval.

The benefits of using this voting system include:

1. Enhanced security: The use of biometric and RFID-based authentication methods ensures that only authorized voters can cast their votes, reducing the risk of fraud or manipulation.

- Scalability: The system can be easily adapted to accommodate different numbers of voters, candidates, or options by modifying the software code.

1. Cost-effectiveness: By leveraging widely available and affordable components, this voting system offers a cost-effective alternative to more complex, commercial solutions.
2. Customization: The system can be tailored to meet the specific needs of different election scenarios by modifying the hardware setup or the software code.
3. Ease of use: The intuitive menu navigation and clear on-screen instructions make the voting process straightforward for both voters and administrators.

Looking ahead, there are several potential improvements and enhancements that could be made to further increase the system's capabilities and adaptability:

1. Adding wireless connectivity: Integrating Wi-Fi or cellular modules to enable remote monitoring and control of the voting process.

2. Expanding authentication options: Incorporating additional authentication methods, such as facial recognition or smart card-based systems.
3. Implementing data encryption: Enhancing the security of vote storage and transmission by encrypting the data using cryptographic techniques.
4. Developing a user-friendly graphical interface: Creating a touchscreen-based interface to improve user experience and simplify menu navigation.
5. Integrating with a web-based platform: Developing a web application to facilitate remote access, real-time monitoring, and data analysis.

By building on the existing foundation and incorporating these improvements, the Arduino-based voting system can continue to evolve and serve as an effective solution for a wide range of voting scenarios.

LINKS TO USEFUL RESOURCES AND REFERENCES

Below is a list of relevant resources and references that that was useful for learning and understanding of the Arduino-based voting system:

Arduino Documentation and Tutorials

- Arduino official website: https://www.arduino.cc/
- Arduino language reference: https://www.arduino.cc/reference/en/
- Arduino playground: https://playground.arduino.cc/

Library Documentation for the Specific Libraries Used in the Project

- Wire library: https://www.arduino.cc/en/reference/wire
- LiquidCrystal_I2C library: https://github.com/johnrickman/LiquidCrystal_I2C
- MFRC522 library (for RFID reader): https://github.com/miguelbalboa/rfid
- Adafruit Fingerprint Sensor library: https://github.com/adafruit/Adafruit-Fingerprint-Sensor-Library
- RTClib (for real-time clock): https://github.com/adafruit/RTClib

Datasheets for the Hardware Components

- RFID reader (MFRC522): https://www.nxp.com/docs/en/datasheet/MFRC522.pdf Fingerprint sensor (e.g., R307): https://www.electronicwings.com/sensors-modules/r307-fingerprint-module

- Real-time clock (e.g., DS3231): https://datasheets.maximintegrated.com/en/ds/DS3231.pdf

External Articles, Blog Posts, or Research Papers Related to the Voting System, its Components, or Similar Projects

- Arduino-based fingerprint voting machine: https://www.instructables.com/Arduino-Fingerprint-Voting-Machine/
- Implementing a simple voting system using Arduino: https://create.arduino.cc/projecthub/ganeshbhat045/arduino-voting-machine-43e485
- Security analysis of electronic voting systems: https://ieeexplore.ieee.org/document/7966943
- An overview of electronic voting systems: https://www.researchgate.net/publication/341120739_An_Overview_of_Electronic_Voting_Systems

Appendices

Complete Code

```
#include <RTClib.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Adafruit_Fingerprint.h>
#include <SoftwareSerial.h>
#include <MFRC522.h>


MFRC522 mfrc522(SS_PIN, RST_PIN);  // Create MFRC522 instance
```

```
SoftwareSerial mySerial(2, 3);
Adafruit_Fingerprint finger = Adafruit_Fingerprint(&mySerial);


LiquidCrystal_I2C lcd(0x27, 16, 2);
// Initialize the DS3231 RTC module
RTC_DS3231 rtc;


const int clkPin = 4;
const int dtPin = 5;
const int swPin = 6;
constexpr uint8_t RST_PIN = 9;  // Configurable, see typical pin layout
constexpr uint8_t SS_PIN = 10;  // Configurable, see typical pin layout
int currentStateCLK;
int previousStateCLK;
int menuOption = 1;
int subMenuOption = 1;
bool inSubMenu = false;
uint8_t id;
bool hasVoted[128] = { false };
uint32_t votes[2] = { 0 };   // Assuming there are 2 candidates
int voteCount[] = { 0, 0 };  // Assuming there are two candidates, initialize
their vote counts to 0


const int MAX_VOTERS = 100;       // Define a constant for the maximum number
of voters
int votedFingerIDs[MAX_VOTERS];  // Declare the votedFingerIDs array
int numVoters = 0;                // Keep track of the number of voters
byte readCard[4];
byte tagID[4];
byte MasterTag[4] = { 0xC2, 0x48, 0xB3, 0x1E };  // Replace with your actual
master tag
unsigned long lastInteraction;
unsigned long idleTimeout = 60000; // 1 minute (60,000 milliseconds)
bool displayingDateTime = false;


void setup() {
  Serial.begin(115200);
  Serial.println("Starting...");

  // Initialize the RFID module
  SPI.begin();         // Init SPI bus
  mfrc522.PCD_Init();  // Init MFRC522
  pinMode(clkPin, INPUT);
  pinMode(dtPin, INPUT);
  pinMode(swPin, INPUT_PULLUP);

  previousStateCLK = digitalRead(clkPin);

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

  finger.begin(57600);
  if (finger.verifyPassword()) {
    lcd.clear();
    lcd.print("Fingerprint");
    lcd.setCursor(0, 1);
    lcd.print("     ACTIVE! :)");
    delay(2000);
  } else {
    lcd.clear();
    lcd.print("Fingerprint");
    lcd.setCursor(0, 1);
    lcd.print("   INACTIVE! :(");
    delay(2000);
  }


}


boolean getID();
void checkResults();


void loop() {
  currentStateCLK = digitalRead(clkPin);
  if (currentStateCLK != previousStateCLK) {
    if (digitalRead(dtPin) != currentStateCLK) {
      if (!inSubMenu) {
        menuOption++;
        if (menuOption > 3) menuOption = 1;
      } else {
        subMenuOption++;
        if (subMenuOption > 2) subMenuOption = 1;
      }
    } else {
      if (!inSubMenu) {
        menuOption--;
        if (menuOption < 1) menuOption = 3;
      } else {
        subMenuOption--;
        if (subMenuOption < 1) subMenuOption = 2;
      }
    }
    updateMenu();
  }
  previousStateCLK = currentStateCLK;


  if (digitalRead(swPin) == LOW) {
    if (!inSubMenu) {
      inSubMenu = true;
      subMenuOption = 1;
      updateMenu();
    } else {
      switch (menuOption) {
        case 1:
          if (subMenuOption == 1) {
            enrollFingerprint();
          } else if (subMenuOption == 2) {
            inSubMenu = false;
          }
          break;
        case 2:
          if (subMenuOption == 1) {
            castVote();  //Add this function later when you have the code.
          } else if (subMenuOption == 2) {
            inSubMenu = false;
          }
          break;
        case 3:
          if (subMenuOption == 1) {
            checkResults();
          } else if (subMenuOption == 2) {
            inSubMenu = false;
          }
          break;
      }
      updateMenu();
    }
    delay(250);
  }
}



void updateMenu() {
  if (!inSubMenu) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("-Enroll");
    lcd.setCursor(10, 0);
    lcd.print("-Vote");
    lcd.setCursor(0, 1);
    lcd.print("-Results");

    switch (menuOption) {
      case 1:
        lcd.setCursor(0, 0);
        lcd.print(">Enroll");
        break;
      case 2:
        lcd.setCursor(10, 0);
        lcd.print(">Vote");
        break;
      case 3:
        lcd.setCursor(0, 1);
        lcd.print(">Results");
        break;
    }
  } else {
    switch (menuOption) {
      case 1:
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("-Enroll FP");
        lcd.setCursor(0, 1);
        lcd.print("-Exit");


        switch (subMenuOption) {
          case 1:
            lcd.setCursor(0, 0);
            lcd.print(">Enroll FP");
            break;
          case 2:
            lcd.setCursor(0, 1);
            lcd.print(">Exit");
            break;
        }
        break;
      case 2:
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("-Cast Vote");
        lcd.setCursor(0, 1);
        lcd.print("-Exit");

        switch (subMenuOption) {
          case 1:
            lcd.setCursor(0, 0);
            lcd.print(">Cast Vote");
            break;
          case 2:
            lcd.setCursor(0, 1);
            lcd.print(">Exit");
            break;
        }
        break;
      case 3:
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("-Check Results");
        lcd.setCursor(0, 1);
        lcd.print("-Exit");

        switch (subMenuOption) {
          case 1:
            lcd.setCursor(0, 0);
            lcd.print(">Check Results");
            break;
          case 2:
            lcd.setCursor(0, 1);
          lcd.print(">Exit");
          break;
        }
        break;
    }
  }
}


void enrollFingerprint() {
  lcd.clear();
  lcd.print("Enroll a fingerprint");
  lcd.setCursor(0, 1);
  lcd.print("Enter ID 1-127");
  delay(3000);


  // Set the initial ID value.
  id = 1;


  // Flag to exit the ID selection loop.
  bool exitIDSelection = false;


  while (!exitIDSelection) {
    lcd.clear();
    lcd.print("ID: ");
    lcd.print(id);
    lcd.setCursor(0, 1);
    lcd.print("Press switch");
    currentStateCLK = digitalRead(clkPin);

    if (currentStateCLK != previousStateCLK) {
      if (digitalRead(dtPin) != currentStateCLK) {
        id++;
        if (id > 127) id = 1;
      } else {
        id--;
        if (id < 1) id = 127;
      }
    }
    previousStateCLK = currentStateCLK;

    if (digitalRead(swPin) == LOW) {
      exitIDSelection = true;
      delay(250);
    }
  }

  lcd.clear();
  lcd.print("Enrolling ID #");
  lcd.print(id);
  delay(3000);

  getFingerprintEnroll();
}
```

```
uint8_t getFingerprintEnroll() {
  lcd.clear();
  lcd.print("Place finger on");
  lcd.setCursor(0, 1);
  lcd.print("sensor...");
  delay(2000);


  // Start of the code copied from the original getFingerprintEnroll()
function
  int p = -1;
  Serial.print("Waiting for valid finger to enroll as #");
  Serial.println(id);
  while (p != FINGERPRINT_OK) {
    p = finger.getImage();
    switch (p) {
      case FINGERPRINT_OK:
        Serial.println("Image taken");


        break;
      case FINGERPRINT_NOFINGER:
        Serial.println(".");
        break;
      case FINGERPRINT_PACKETRECIEVEERR:
        Serial.println("Communication error");
        lcd.clear();
        lcd.print("Comm. error");
        delay(2000);
        break;
      case FINGERPRINT_IMAGEFAIL:
        Serial.println("Imaging error");
        lcd.clear();
        lcd.print("Imaging error");
        delay(2000);
        break;
      default:
        Serial.println("Unknown error");
        lcd.clear();
        lcd.print("Unknown error");
        delay(2000);
        break;
    }
  }

  // OK success!

  p = finger.image2Tz(1);
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image converted");
      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");
      lcd.clear();
      lcd.print("No finger Features");
      lcd.setCursor(0, 1);
      lcd.print("Enroll Again!");
      delay(2000);
      return p;
    case FINGERPRINT_INVALIDIMAGE:
      Serial.println("Could not find fingerprint features");
      lcd.clear();
      lcd.print("No finger Features");
      lcd.setCursor(0, 1);
      lcd.print("Enroll Again!");
      delay(2000);
      return p;
    default:
      Serial.println("Unknown error");
      return p;
  }


  Serial.println("Remove finger");
  lcd.clear();
  lcd.print("Remove Finger");


  delay(2000);
  p = 0;
  while (p != FINGERPRINT_NOFINGER) {
    p = finger.getImage();
  }
  Serial.print("ID ");
  Serial.println(id);
```

```
p = -1;
Serial.println("Place same finger again");
lcd.clear();
lcd.print("Place same");
lcd.setCursor(0, 1);
lcd.print("Finger Again!");


while (p != FINGERPRINT_OK) {
  p = finger.getImage();
  switch (p) {
    case FINGERPRINT_OK:
      Serial.println("Image taken");
      break;
    case FINGERPRINT_NOFINGER:
      Serial.print(".");
      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");
    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");
    lcd.clear();
    lcd.print("No finger Features");
    lcd.setCursor(0, 1);
    lcd.print("Enroll Again!");
    delay(2000);
    return p;
  case FINGERPRINT_INVALIDIMAGE:
    Serial.println("Could not find fingerprint features");
    lcd.clear();
    lcd.print("No finger Features");
    lcd.setCursor(0, 1);
    lcd.print("Enroll Again!");
    delay(2000);
    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!");
} 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!");
  lcd.clear();
  lcd.print("Fingerprint ID");
  lcd.setCursor(0, 1);
```

```
    lcd.print(id);
    lcd.print(" enrolled!");
    delay(3000);
  } 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;
  }

  return true;
}

void castVote() {
  int fingerID;

  // Scan the fingerprint
  lcd.clear();
  lcd.print("Scan your finger");
  delay(2000);

  fingerID = getFingerprintIDez();
```

```
delay(2000);
if (fingerID >= 0) {
  if (hasAlreadyVoted(fingerID)) {
    lcd.clear();
    lcd.print("Already voted");
    delay(2000);
  } else {
    // Show the candidates and select one using the encoder
    int selectedCandidate = selectCandidateUsingEncoder();

    // Update the vote count for the selected candidate
    voteCount[selectedCandidate - 1]++;

    // Record the fingerID that has voted
    if (numVoters < MAX_VOTERS) {
      votedFingerIDs[numVoters++] = fingerID;
    }

    lcd.clear();
    lcd.print("Vote cast for");
    lcd.setCursor(0, 1);
    lcd.print("Candidate ");
    lcd.print(selectedCandidate);
    delay(3000);
  }
} else {
  lcd.clear();
  lcd.print("Finger Not Seen!");
  delay(2000);
  }
}



bool hasAlreadyVoted(int fingerID) {
  for (int i = 0; i < numVoters; i++) {
    if (votedFingerIDs[i] == fingerID) {
      return true;
    }
  }
  return false;
}



int getFingerprintIDez() {
  uint8_t p = finger.getImage();
  if (p != FINGERPRINT_OK) return -1;

  p = finger.image2Tz();
  if (p != FINGERPRINT_OK) return -1;

  p = finger.fingerFastSearch();
  if (p != FINGERPRINT_OK) return -1;

  // Found a match!
  Serial.print("Found ID #");
  Serial.print(finger.fingerID);
  Serial.print(" with confidence of ");
  Serial.println(finger.confidence);


  return finger.fingerID;
}



int selectCandidateUsingEncoder() {
  int selectedCandidate = 1;  // Default selection is Candidate A
  bool selectionConfirmed = false;


  while (!selectionConfirmed) {
    currentStateCLK = digitalRead(clkPin);
    if (currentStateCLK != previousStateCLK) {
      if (digitalRead(dtPin) != currentStateCLK) {
        selectedCandidate++;
        if (selectedCandidate > 1) selectedCandidate = 0;
      } else {
        selectedCandidate--;
        if (selectedCandidate < 0) selectedCandidate = 1;
      }
      updateCandidateSelection(selectedCandidate);
    }
    previousStateCLK = currentStateCLK;


    if (digitalRead(swPin) == LOW) {
      selectionConfirmed = true;
      delay(250);
    }
  }

  return selectedCandidate;
}

void updateCandidateSelection(int selectedCandidate) {
  lcd.clear();
  if (selectedCandidate == 0) {
    lcd.print(">1. Candidate 1");
    lcd.setCursor(0, 1);
    lcd.print(" 2. Candidate 2");
  } else {
    lcd.print(" 1. Candidate 1");
    lcd.setCursor(0, 1);
    lcd.print(">2. Candidate 2");
  }
}



void checkResults() {

  lcd.clear();
  lcd.print("Swipe card");
  delay(1000);

  bool cardDetected = false;
  while (!cardDetected) {
    if (getID()) {
      cardDetected = true;
      lcd.clear();
      lcd.setCursor(0, 0);

      if (memcmp(tagID, MasterTag, 4) == 0) {
        lcd.print(" Access Granted!");
        lcd.clear();
        lcd.print("Candidate 1: ");
        lcd.print(voteCount[0]);
        lcd.setCursor(0, 1);
        lcd.print("Candidate 2: ");
        lcd.print(voteCount[1]);
        delay(5000);
      } else {
        lcd.print(" Access Denied!");
      }
      delay(2000);
    }
  }
}


boolean getID() {
  // Getting ready for Reading PICCs
  if (!mfrc522.PICC_IsNewCardPresent()) {  // If a new PICC placed to RFID
reader continue
    return false;
  }
  if (!mfrc522.PICC_ReadCardSerial()) {  // Since a PICC placed get Serial and
continue
    return false;
  }

  for (uint8_t i = 0; i < 4; i++) {  // The MIFARE PICCs that we use have 4
byte UID
    tagID[i] = mfrc522.uid.uidByte[i];
  }

  // Print the UID in HEX format
  for (uint8_t i = 0; i < 4; i++) {
    Serial.print(tagID[i], HEX);
    if (i < 3) {
      Serial.print(' ');  // Add a space between bytes
    } else {
      Serial.println();  // Add a newline after the last byte
    }
  }

  mfrc522.PICC_HaltA();  // Stop reading
  return true;
}
```