ESP32 Code

```
#include <WiFi.h>
#include <WebSocketsServer.h>
#include <ESP32Servo.h>
#include <time.h>
#include <DHT.h>
#include <SPI.h>
#include <TFT_eSPI.h>
#include "FS.h"
#include "DFRobotDFPlayerMini.h"

HardwareSerial mySerial(2);
DFRobotDFPlayerMini player;
// TFT display
TFT_eSPI tft = TFT_eSPI();

// DHT sensor pins and setup
#define DHTPIN 22 // Pin which is connected to the DHT sensor
#define DHTTYPE DHT11 // DHT 11 or DHT22, depending on your sensor
DHT dht(DHTPIN, DHTTYPE);

// Button properties
#define BUTTON_WIDTH 200
#define BUTTON_HEIGHT 100

#define BUTTON1_X (480 / 4 - BUTTON_WIDTH / 2) // First button centered in the
left half
#define BUTTON1_Y (320 - BUTTON_HEIGHT - 20) // A bit higher than the bottom
#define BUTTON2_X (480 * 3 / 4 - BUTTON_WIDTH / 2) // Second button centered in
the right half
#define BUTTON2_Y (320 - BUTTON_HEIGHT - 20) // A bit higher than the bottom

bool button1State = false; // Will be used to determine the binary state
bool button2State = false;

// Create buttons
TFT_eSPI_Button button1;
TFT_eSPI_Button button2;

// Calibration file and flag
#define CALIBRATION_FILE "/TouchCalData"
#define REPEAT_CAL false

int smokeSensorPin = 33;
```

```
int smokeThreshold = 2000;

const char* ssid = "Your-SSID";
const char* password = "Your-Password";



WebSocketsServer webSocket = WebSocketsServer(8080); // WebSocket server on port
8080
Servo DoorServo; // Create a servo object
Servo FeederServo;
int CurrentHour;
int counter = 0;
int CurrentMinute;
int DoorServoPin = 32;
int FeederServoPin = 14;
bool doorOpen = false;
bool audioEnabled = false;
bool audioState = false;
float temperature = 25.5; // Example temperature value
float humidity = 50.0; // Example humidity value
bool emergencyStatus = false; // Example emergency status

// Define the start and end positions
const int FeederStartPos = 0;
const int FeederEndPos = 40;
const int FeederStepDelay = 45;
const int FeederNumRepeats = 3;
int FeederHour = 0;
int FeederMinute = 0;

unsigned long lastTimeUpdate = 0; // Last time update in milliseconds
unsigned long lastTimeUpdate2 = 0;
const unsigned long updateInterval = 60000; // Update interval of 1 minute (60000
milliseconds)

void updateTime() {
  // Get the current time
  time_t now;
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to obtain time");
    return;
  }
  CurrentHour = timeinfo.tm_hour;
```

```
  Serial.println(CurrentHour);
  CurrentMinute = timeinfo.tm_min;
  Serial.println(CurrentMinute);
  if (FeederHour == CurrentHour && FeederMinute == CurrentMinute){
    RunFeeder();
  }
}

void RunFeeder(){
  Serial.println("THE FEEDER IS RUNNING");
  // Perform the movement sequence numRepeats times
  for (int repeat = 0; repeat < FeederNumRepeats; repeat++) {
    // Move the servo from 0 to 40
    for (int pos = FeederStartPos; pos <= FeederEndPos; pos++) {
      FeederServo.write(pos);
      delay(FeederStepDelay);
    }

    // Move the servo from 40 back to 0
    for (int pos = FeederEndPos; pos >= FeederStartPos; pos--) {
      FeederServo.write(pos);
      delay(FeederStepDelay);
    }

    // Wait for 1 seconds between cycles
    delay(1000);
  }

  // After completing all cycles, wait at position 0 for 10 seconds
  FeederServo.write(FeederStartPos);
  delay(15000);
}

void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length)
{
  switch(type) {
    case WStype_DISCONNECTED:
      Serial.printf("[%u] Disconnected!\n", num);
      break;
    case WStype_CONNECTED: {
      IPAddress ip = webSocket.remoteIP(num);
      Serial.printf("[%u] Connected from %d.%d.%d.%d url: %s\n", num, ip[0],
ip[1], ip[2], ip[3], payload);
      // Send initial state on connection
      webSocket.sendTXT(num, doorOpen ? "Door Open" : "Door Closed");
```

```
    }
    break;
    case WStype_TEXT:
      Serial.printf("[%u] Received text: %s\n", num, payload);

      // Toggle door state if "toggle" message received
      if (strcmp((const char*)payload, "toggle") == 0) {
        doorOpen = !doorOpen;
        webSocket.broadcastTXT(doorOpen ? "Door Open" : "Door Closed");

        // Control servo based on door state
        if (doorOpen) {
          DoorServo.write(90); // Move servo to 90 degrees when door is open
          button1State = true;
          drawButton1();
        } else {
          DoorServo.write(0); // Move servo to 0 degrees when door is closed
          button1State = false;
          drawButton1();
        }
      }

      // Send additional data if "ping" message received
      if (strcmp((const char*)payload, "ping") == 0) {




        char message[100];
        snprintf(message, sizeof(message), "Door: %s, Temp: %.1f F, Humidity:
%.1f%%, Emergency: %s, Audio: %s",
                 doorOpen ? "Door Open" : "Door Closed", temperature, humidity,
emergencyStatus ? "Active" : "Inactive", audioState ? "Playing" : "Paused");
        webSocket.sendTXT(num, message);
      }

      if (strcmp((const char*)payload, "feed") == 0) {
        RunFeeder();
      }



      if (strcmp((const char*)payload, "play") == 0 && audioEnabled) {
        button2State = true;
          drawButton2();
```

```
        player.play(1);
         Serial.println("Play Audio");
        audioState = true;

      }

if (strncmp((const char*)payload, "volume:", 7) == 0) {
        int newVolume;
        if (sscanf((const char*)payload + 7, "%d", &newVolume) == 1) {
          Serial.printf("Setting volume to %d\n", newVolume);
          // Set the volume on the audio player (replace with your specific
method)
          player.volume(newVolume);
        } else {
          Serial.println("Invalid volume format");
        }
      }

      if (strcmp((const char*)payload, "pause") == 0 && audioEnabled)  {
        player.pause();
        audioState = false;
        Serial.println("Pause Audio");
        button2State = false;
          drawButton2();
      }

      if (strncmp((const char*)payload, "Time ", 5) == 0) {
        int feederHour;
        int feederMinute;
        if (sscanf((const char*)payload + 5, "%d:%d", &feederHour, &feederMinute)
== 2) {
          FeederHour = feederHour;
          FeederMinute = feederMinute;
          Serial.printf("Feeder time set to %02d:%02d\n", FeederHour,
FeederMinute);
        } else {
          Serial.println("Invalid time format");
        }
      }

      break;
  }
}
void setup() {
  Serial.begin(9600);
```

```
  pinMode(smokeSensorPin, INPUT);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // Print the IP address
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());

  // Initialize time
  configTime(-5 * 3600, 3600, "pool.ntp.org", "time.nist.gov"); // Set timezone
to Eastern Standard Time and DST offset

  webSocket.begin();
  webSocket.onEvent(webSocketEvent);

  // Update the time initially
  updateTime();
  lastTimeUpdate = millis();
  lastTimeUpdate2 = millis();

  DoorServo.setPeriodHertz(50); // Set PWM frequency for SG90 servo (default is
50Hz)
  DoorServo.attach(DoorServoPin); // Attach the servo to the specified pin with
min and max pulse width in microseconds

  FeederServo.setPeriodHertz(50); // Set PWM frequency for SG90 servo (default is
50Hz)
  FeederServo.attach(FeederServoPin);
  FeederServo.write(FeederStartPos);

  Serial.println("WebSocket server started");

delay(1000);
  // Initialize SPIFFS
  if (!SPIFFS.begin(true)) {
    Serial.println("SPIFFS initialization failed!");
    //while (1);
  }

  // Initialize TFT display
  tft.init();
```

```
    Serial.println("Test6");
  tft.setRotation(1); // Landscape mode
  tft.fillScreen(TFT_BLACK);
  tft.setTextSize(2);
  Serial.println("Test7");
  // Calibrate touch screen
  touch_calibrate();
  Serial.println("Test8");
  // Draw initial buttons
  drawButtons();
  Serial.println("Test9");
  mySerial.begin(9600, SERIAL_8N1, 27, 26); // RX = GPIO 27, TX = GPIO 26
Serial.println("Test2");

  Serial.println("Initializing DFPlayer Mini...");

  // Start communication with DFPlayer Mini
  if (player.begin(mySerial)) {
    Serial.println("DFPlayer Mini online.");
    audioEnabled = true;
    // Set volume to a reasonable level (0 to 30)
    player.volume(30);
    // Play the first MP3 file on the SD card
    player.play(1);
  } else {
    Serial.println("Connecting to DFPlayer Mini failed!");
    Serial.println("Please check the connection and SD card.");
  }
  button2State = false;
          drawButton2();
  delay(1000);
  dht.begin();

}

void loop() {

     int sensorValue = analogRead(smokeSensorPin);
     Serial.println(analogRead(smokeSensorPin));
  if (sensorValue < 3200){
    DoorServo.write(90);
    button1State = true;
    doorOpen = true;
    emergencyStatus = true;
  } else {
```

```
    emergencyStatus = false;
  }

   humidity = dht.readHumidity();
        temperature = (dht.readTemperature() * 1.8) + 32;
         if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    tft.setCursor(0, 200);
    tft.setTextColor(TFT_RED);
    tft.println("Sensor Error");
    delay(1000);
    return;
  }

  uint16_t x, y;
  bool pressed = tft.getTouch(&x, &y);
  webSocket.loop();


  if (millis() - lastTimeUpdate >= updateInterval) {
    updateTime(); // Update the current time
    player.pause();
    lastTimeUpdate = millis(); // Reset the timer
  }

 if (millis() - lastTimeUpdate2 >= 5000){
   updateTemperatureHumidity(temperature, humidity);
   lastTimeUpdate2 = millis();

 }

//updateTemperatureHumidity(temperature, humidity);
  if (pressed) {
    if (button1.contains(x, y)) {
      button1.press(true);  // Tell the button it is pressed
    } else {
      button1.press(false);  // Tell the button it is NOT pressed
    }

    if (button2.contains(x, y)) {
      button2.press(true);  // Tell the button it is pressed
    } else {
      button2.press(false);  // Tell the button it is NOT pressed
    }
  } else {
```

```
    button1.press(false);  // Tell the button it is NOT pressed
    button2.press(false);  // Tell the button it is NOT pressed
  }

  // Check if the buttons have changed state
  if (button1.justReleased()) button1.drawButton();  // Draw normal
  if (button1.justPressed()) {
    button1State = !button1State;
    drawButton1();
  }

  if (button2.justReleased()) button2.drawButton();  // Draw normal
  if (button2.justPressed()) {
    button2State = !button2State;
    drawButton2();
  }

  delay(15); // Small delay for debouncing

  // Logic to control the door based on button1 state
  if (button1State) {
    doorOpen = true;
  } else {
    doorOpen = false;
  }

  if (doorOpen) {
    DoorServo.write(90);
  } else {
    DoorServo.write(0);
  }
  if(button2State && audioEnabled && !audioState){
       player.play(1);
         Serial.println("Play Audio (screen)");
        audioState = true;
 } else if (!button2State && audioEnabled && audioState) {
  player.pause();
         Serial.println("Pause Audio (screen)");
        audioState = false;
 }
}



void drawButtons() {
```

```
  drawButton1();
  drawButton2();
}

void drawButton1() {
  uint16_t color = button1State ? TFT_RED : TFT_GREEN;
  button1.initButton(&tft, BUTTON1_X + BUTTON_WIDTH / 2, BUTTON1_Y +
BUTTON_HEIGHT / 2, BUTTON_WIDTH, BUTTON_HEIGHT, TFT_WHITE, color, TFT_WHITE,
"Door", 2);
  tft.setTextColor(TFT_WHITE); // Set text color for the button label
  tft.setTextSize(2); // Set text size for the button label
  button1.drawButton();
}

void drawButton2() {
  uint16_t color = button2State ? TFT_RED : TFT_GREEN;
  char label[6];  // Allocate space for the label
  strcpy(label, button2State ? "Pause" : "Play");  // Copy the appropriate label
into the array
  button2.initButton(&tft, BUTTON2_X + BUTTON_WIDTH / 2, BUTTON2_Y +
BUTTON_HEIGHT / 2, BUTTON_WIDTH, BUTTON_HEIGHT, TFT_WHITE, color, TFT_WHITE,
label, 2);
  tft.setTextColor(TFT_WHITE); // Set text color for the button label
  tft.setTextSize(2); // Set text size for the button label
  button2.drawButton();
}

void touch_calibrate() {
  uint16_t calData[5];
  uint8_t calDataOK = 0;

  // Check if calibration file exists and size is correct
  if (SPIFFS.exists(CALIBRATION_FILE)) {
    if (REPEAT_CAL) {
      // Delete if we want to re-calibrate
      SPIFFS.remove(CALIBRATION_FILE);
    } else {
      fs::File f = SPIFFS.open(CALIBRATION_FILE, "r");
      if (f) {
        if (f.readBytes((char *)calData, 14) == 14) {
          calDataOK = 1;
        }
        f.close();
      }
    }
```

```
  }

  if (calDataOK && !REPEAT_CAL) {
    // Calibration data valid
    tft.setTouch(calData);
  } else {
    // Data not valid so recalibrate
    tft.fillScreen(TFT_BLACK);
    tft.setCursor(20, 0);
    tft.setTextFont(2);
    tft.setTextSize(1);
    tft.setTextColor(TFT_WHITE, TFT_BLACK);

    tft.println("Touch corners as indicated");

    tft.setTextFont(1);
    tft.println();

    if (REPEAT_CAL) {
      tft.setTextColor(TFT_RED, TFT_BLACK);
      tft.println("Set REPEAT_CAL to false to stop this running again!");
    }

    tft.calibrateTouch(calData, TFT_MAGENTA, TFT_BLACK, 15);

    // Store data
    fs::File f = SPIFFS.open(CALIBRATION_FILE, "w");
    if (f) {
      f.write((const unsigned char *)calData, 14);
      f.close();
    }

    // Clear the screen after calibration
    tft.fillScreen(TFT_BLACK);
  }
}

void clearButtons() {
  tft.fillRect(BUTTON1_X, BUTTON1_Y, BUTTON_WIDTH, BUTTON_HEIGHT, TFT_BLACK);
  tft.fillRect(BUTTON2_X, BUTTON2_Y, BUTTON_WIDTH, BUTTON_HEIGHT, TFT_BLACK);
}

void updateTemperatureHumidity(float temperature, float humidity) {
  // Clear previous readings
  tft.fillRect(0, 0, 480, 100, TFT_BLACK);
```

```
  // Display temperature
  tft.setTextColor(TFT_CYAN, TFT_BLACK);
  tft.setTextDatum(TC_DATUM);
  tft.setTextFont(4);
  tft.drawString("Temp:", 120, 10);
  tft.setTextFont(6);
  tft.drawString(String(temperature, 1) + " C", 120, 60); // Adjusted y-
coordinate

  // Display humidity
  tft.setTextColor(TFT_YELLOW, TFT_BLACK);
  tft.setTextFont(4);
  tft.drawString("Humidity:", 360, 10);
  tft.setTextFont(6);
  tft.drawString(String(humidity, 1) + " %", 360, 60); // Adjusted y-coordinate

  // Redraw buttons to ensure they are displayed correctly
    tft.setTextFont(2);
  clearButtons();
  drawButtons();
}
```