LAMPIRAN

Dokumentasi Penelitian

Sketch yang digunakan

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include <Adafruit_TCS34725.h>
#include <LiquidCrystal_I2C.h>
#include <Keypad.h>
#include <ArduinoJson.h>
#include <Preferences.h>

// ============================================
// KONFIGURASI WIFI
// ============================================
#define WIFI_SSID "WAHYU"
```

```
#define WIFI_PASSWORD "12345678"

// ==========================================
// KONFIGURASI MQTT
// ==========================================
#define MQTT_BROKER "broker.hivemq.com"
#define MQTT_PORT 1883
#define MQTT_CLIENT_ID "esp32-piggybank-01"
#define DEVICE_ID "piggybank-01"

#define TOPIC_MONEY       "piggybank/money"
#define TOPIC_PIN         "piggybank/pin"
#define TOPIC_STATUS      "piggybank/status"
#define TOPIC_UNLOCK      "piggybank/unlock"
#define TOPIC_RESET       "piggybank/reset"
#define TOPIC_ACTIVITY    "piggybank/activity"
#define TOPIC_DEVICE      "piggybank/device"
#define TOPIC_CALIBRATION "piggybank/calibration"
#define TOPIC_CONFIG      "piggybank/config"

// ==========================================
// PIN MAPPING
// ==========================================
#define I2C_SDA 21
#define I2C_SCL 22

#define LCD_ADDRESS 0x27
#define LCD_COLS 16
#define LCD_ROWS 2

#define RELAY_PIN 17
#define RELAY_ACTIVE_HIGH true

#define LED_HIJAU_PIN 18
#define LED_MERAH_PIN 19

#define BUZZER_PIN 23

#define KEYPAD_ROWS 4
#define KEYPAD_COLS 4
byte keypadRowPins[KEYPAD_ROWS] = {32, 33, 25, 26};
byte keypadColPins[KEYPAD_COLS] = {27, 14, 4, 16};

char keypadKeys[KEYPAD_ROWS][KEYPAD_COLS] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
```

```
    {'*', '0', '#', 'D'}
};

// ==========================================
// OBJEK GLOBAL
// ==========================================
WiFiClient espClient;
PubSubClient mqttClient(espClient);
Adafruit_TCS34725 tcs =
Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);
LiquidCrystal_I2C lcd(LCD_ADDRESS, LCD_COLS, LCD_ROWS);
Keypad keypad = Keypad(makeKeymap(keypadKeys), keypadRowPins,
keypadColPins, KEYPAD_ROWS, KEYPAD_COLS);
Preferences prefs;

// ==========================================
// STATE KALIBRASI
// ==========================================
struct Calibration {
    int nominal;
    int r, g, b;
    int threshold;
    bool valid;
};
Calibration calibrationData[5];
int nominalList[5] = {5000, 10000, 20000, 50000, 100000};

// ==========================================
// STATE INPUT PIN & MODE
// ==========================================
String pinBuffer = "";
bool calibrationMode = false;
int calibrationTargetIndex = -1;
bool readingMode = false;

// ==========================================
// STATE KONFIRMASI DETEKSI UANG
// ==========================================
bool confirmationMode = false;
int pendingNominal = 0;
unsigned long confirmationStartAt = 0;
const unsigned long CONFIRMATION_TIMEOUT_MS = 10000; // 10 detik,
auto batal kalau tidak dijawab

// ==========================================
// STATE TOTAL TABUNGAN (dari server via TOPIC_STATUS)
// ==========================================
```

```
long cachedTotalTabungan = -1;

// ============================================
// STATE UNLOCK (relay timer non-blocking)
// ============================================
bool relayActive = false;
unsigned long relayActivatedAt = 0;
const unsigned long RELAY_DURATION_MS = 3000;

// ============================================
// STATE PEMBACAAN SENSOR
// ============================================
unsigned long lastSensorCheck = 0;
const unsigned long SENSOR_CHECK_INTERVAL_MS = 500;
unsigned long lastDetectedAt = 0;
const unsigned long DETECT_COOLDOWN_MS = 3000;


// ============================================
// SETUP AWAL
// ============================================
void setup() {
    Serial.begin(115200);
    delay(500);

    pinMode(RELAY_PIN, OUTPUT);
    pinMode(LED_HIJAU_PIN, OUTPUT);
    pinMode(LED_MERAH_PIN, OUTPUT);
    pinMode(BUZZER_PIN, OUTPUT);
    relayWrite(false);
    digitalWrite(LED_HIJAU_PIN, LOW);
    digitalWrite(LED_MERAH_PIN, LOW);
    digitalWrite(BUZZER_PIN, LOW);

    Wire.begin(I2C_SDA, I2C_SCL);

    lcd.init();
    lcd.backlight();
    lcdShowMessage("Booting...", "");

    if (tcs.begin()) {
        Serial.println("[SENSOR] TCS34725 terdeteksi");
    } else {
        Serial.println("[SENSOR] TCS34725 TIDAK terdeteksi, cek
wiring!");
        lcdShowMessage("Sensor Error", "Cek wiring");
        buzzerBeep(3, 200);
```

```
    }

    for (int i = 0; i < 5; i++) {
        calibrationData[i].nominal = nominalList[i];
        calibrationData[i].valid = false;
    }

    loadCalibrationFromFlash();

    wifiInit();

    mqttClient.setServer(MQTT_BROKER, MQTT_PORT);
    mqttClient.setCallback(mqttCallback);
    mqttClient.setBufferSize(1024);
    mqttConnect();

    showIdleScreen();
}


// ==========================================
// LOOP UTAMA
// ==========================================
void loop() {
    wifiReconnectIfNeeded();

    if (!mqttClient.connected()) {
        mqttConnect();
    }
    mqttClient.loop();

    handleKeypad();
    handleColorSensor();
    handleRelayTimer();
    handleConfirmationTimeout();

    delay(20);
}


// ==========================================
// WIFI
// ==========================================
void wifiInit() {
    Serial.print("[WIFI] Menghubungkan ke: ");
    Serial.println(WIFI_SSID);
    WiFi.mode(WIFI_STA);
```

```
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

    int attempt = 0;
    while (WiFi.status() != WL_CONNECTED && attempt < 30) {
        delay(500);
        Serial.print(".");
        attempt++;
    }

    if (WiFi.status() == WL_CONNECTED) {
        Serial.println();
        Serial.print("[WIFI] Terhubung! IP: ");
        Serial.println(WiFi.localIP());
    } else {
        Serial.println();
        Serial.println("[WIFI] Gagal konek, akan dicoba lagi di
loop()");
    }
}

void wifiReconnectIfNeeded() {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("[WIFI] Terputus, mencoba reconnect...");
        WiFi.disconnect();
        WiFi.reconnect();
        delay(1000);
    }
}


// ==========================================
// MQTT
// ==========================================
void mqttConnect() {
    while (!mqttClient.connected() && WiFi.status() == WL_CONNECTED)
{
        Serial.print("[MQTT] Menghubungkan ke broker...");
        if (mqttClient.connect(MQTT_CLIENT_ID)) {
            Serial.println(" berhasil!");
            mqttClient.subscribe(TOPIC_UNLOCK);
            mqttClient.subscribe(TOPIC_CONFIG);
            mqttClient.subscribe(TOPIC_RESET);
            mqttClient.subscribe(TOPIC_STATUS);
            publishDeviceStatus("online");
        } else {
            Serial.print(" gagal, rc=");
            Serial.print(mqttClient.state());
```

```
            Serial.println(" coba lagi 3 detik...");
            delay(3000);
        }
    }
}

void mqttCallback(char* topic, byte* payload, unsigned int length) {
    String message;
    for (unsigned int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    Serial.print("[MQTT] Pesan masuk [");
    Serial.print(topic);
    Serial.print("] (");
    Serial.print(length);
    Serial.print(" byte): ");
    Serial.println(message);

    String topicStr = String(topic);

    if (topicStr == TOPIC_UNLOCK) {
        handleUnlockMessage(message);
    } else if (topicStr == TOPIC_CONFIG) {
        handleConfigMessage(message);
    } else if (topicStr == TOPIC_STATUS) {
        handleStatusMessage(message);
    }
}

void publishJson(const char* topic, JsonDocument& doc) {
    char buffer[512];
    serializeJson(doc, buffer);
    mqttClient.publish(topic, buffer);
}

void publishDeviceStatus(const char* status) {
    JsonDocument doc;
    doc["device_id"] = DEVICE_ID;
    doc["status"] = status;
    doc["firmware_version"] = "1.0.0";
    doc["wifi_rssi"] = WiFi.RSSI();
    publishJson(TOPIC_DEVICE, doc);
}

void publishActivity(const char* activityType, const char*
description) {
    JsonDocument doc;
```

```
    doc["device_id"] = DEVICE_ID;
    doc["activity_type"] = activityType;
    doc["description"] = description;
    publishJson(TOPIC_ACTIVITY, doc);
}


// ==========================================
// HANDLER: Pesan Unlock / Denied dari Server
// ==========================================
void handleUnlockMessage(String message) {
    JsonDocument doc;
    DeserializationError error = deserializeJson(doc, message);
    if (error) return;

    String status = doc["status"] | "";

    if (status == "unlock") {
        activateRelay();
        lcdShowMessage("Terbuka!", "Silakan ambil");
        ledBlinkHijau();
        delay(2000);
        showIdleScreen();
    } else if (status == "denied") {
        lcdShowMessage("PIN Salah!", "Coba lagi");
        ledBlinkMerah();
        buzzerBeep(2, 150);
        delay(2000);
        showIdleScreen();
    }
}


// ==========================================
// HANDLER: Total Tabungan dari Server
// ==========================================
void handleStatusMessage(String message) {
    JsonDocument doc;
    DeserializationError error = deserializeJson(doc, message);
    if (error) return;

    if (doc["total_tabungan"].is<long>() ||
doc["total_tabungan"].is<int>()) {
        cachedTotalTabungan = doc["total_tabungan"].as<long>();
        Serial.print("[STATUS] Total tabungan diperbarui (tersimpan,
tidak ditampilkan otomatis): Rp");
        Serial.println(cachedTotalTabungan);
```

```
    }
}


// ========================================
// HANDLER: Config Kalibrasi dari Server
// ========================================
void handleConfigMessage(String message) {
    JsonDocument doc;
    DeserializationError error = deserializeJson(doc, message);
    if (error) {
        Serial.print("[CONFIG] Gagal parse JSON: ");
        Serial.println(error.c_str());
        return;
    }

    if (doc["calibration"].is<JsonArray>()) {
        JsonArray arr = doc["calibration"];
        for (JsonObject item : arr) {
            int nominal = item["nominal"].as<int>();
            for (int i = 0; i < 5; i++) {
                if (calibrationData[i].nominal == nominal) {
                    calibrationData[i].r = item["r_value"];
                    calibrationData[i].g = item["g_value"];
                    calibrationData[i].b = item["b_value"];
                    calibrationData[i].threshold =
item["threshold"];
                    calibrationData[i].valid = true;
                }
            }
        }
        Serial.println("[CONFIG] Kalibrasi diperbarui dari
server:");

        for (int i = 0; i < 5; i++) {
            Serial.print("  Nominal ");
Serial.print(calibrationData[i].nominal);
            Serial.print(" | valid: ");
Serial.print(calibrationData[i].valid ? "YA" : "TIDAK");
            Serial.print(" | R:");
Serial.print(calibrationData[i].r);
            Serial.print(" G:"); Serial.print(calibrationData[i].g);
            Serial.print(" B:"); Serial.print(calibrationData[i].b);
            Serial.print(" Threshold:");
Serial.println(calibrationData[i].threshold);
        }
```

```
        saveCalibrationToFlash();

        lcdShowMessage("Kalibrasi", "Diperbarui");
        delay(1000);
        showIdleScreen();
    }
}


// ==========================================
// PENYIMPANAN KALIBRASI KE FLASH (Preferences)
// ==========================================
void saveCalibrationToFlash() {
    prefs.begin("calib", false);
    for (int i = 0; i < 5; i++) {
        String prefix = "c" + String(i) + "_";
        prefs.putInt((prefix + "r").c_str(), calibrationData[i].r);
        prefs.putInt((prefix + "g").c_str(), calibrationData[i].g);
        prefs.putInt((prefix + "b").c_str(), calibrationData[i].b);
        prefs.putInt((prefix + "th").c_str(),
calibrationData[i].threshold);
        prefs.putBool((prefix + "v").c_str(),
calibrationData[i].valid);
    }
    prefs.end();
    Serial.println("[FLASH] Kalibrasi disimpan ke memori internal
ESP32");
}

void loadCalibrationFromFlash() {
    prefs.begin("calib", true);
    bool adaData = false;

    for (int i = 0; i < 5; i++) {
        String prefix = "c" + String(i) + "_";
        bool v = prefs.getBool((prefix + "v").c_str(), false);
        if (v) {
            calibrationData[i].r = prefs.getInt((prefix +
"r").c_str(), 0);
            calibrationData[i].g = prefs.getInt((prefix +
"g").c_str(), 0);
            calibrationData[i].b = prefs.getInt((prefix +
"b").c_str(), 0);
            calibrationData[i].threshold = prefs.getInt((prefix +
"th").c_str(), 40);
            calibrationData[i].valid = true;
            adaData = true;
```

```
        }
    }
    prefs.end();

    if (adaData) {
        Serial.println("[FLASH] Kalibrasi lama ditemukan di memori
internal, langsung dipakai:");
        for (int i = 0; i < 5; i++) {
            Serial.print("  Nominal ");
Serial.print(calibrationData[i].nominal);
            Serial.print(" | valid: ");
Serial.print(calibrationData[i].valid ? "YA" : "TIDAK");
            Serial.print(" | R:");
Serial.print(calibrationData[i].r);
            Serial.print(" G:"); Serial.print(calibrationData[i].g);
            Serial.print(" B:");
Serial.println(calibrationData[i].b);
        }
    } else {
        Serial.println("[FLASH] Belum ada kalibrasi tersimpan,
menunggu data dari server...");
    }
}


// ==========================================
// SENSOR WARNA — deteksi nominal (dengan konfirmasi)
// ==========================================
void handleColorSensor() {
    if (calibrationMode) return;
    if (confirmationMode) return; // sedang menunggu jawaban 0/1,
jangan baca sensor dulu

    unsigned long now = millis();
    if (now - lastSensorCheck < SENSOR_CHECK_INTERVAL_MS) return;
    lastSensorCheck = now;

    uint16_t r, g, b, c;
    tcs.getRawData(&r, &g, &b, &c);

    if (c < 100) {
        if (readingMode) lcdShowMessage("Tidak ada obj", "Dekatkan
uang");
        return;
    }

    int rNorm = (int)((long)r * 1000 / c);
```

```
    int gNorm = (int)((long)g * 1000 / c);
    int bNorm = (int)((long)b * 1000 / c);

    long distance;
    int closestNominal;
    int matchedNominal = matchNominal(rNorm, gNorm, bNorm, distance,
closestNominal);

    Serial.print("[DEBUG] Rn:"); Serial.print(rNorm);
    Serial.print(" Gn:"); Serial.print(gNorm);
    Serial.print(" Bn:"); Serial.print(bNorm);
    Serial.print(" | Terdekat: Rp"); Serial.print(closestNominal);
    Serial.print(" jarak:"); Serial.println(distance);

    if (readingMode) {
        // FIX: buffer diperbesar dari 17 -> 24 supaya tidak
overflow saat rNorm/gNorm/bNorm 4 digit
        char line1[24], line2[24];
        sprintf(line1, "R%d G%d B%d", rNorm, gNorm, bNorm);
        sprintf(line2, "Rp%d j:%ld", closestNominal, distance);
        lcdShowMessage(String(line1), String(line2));
        return;
    }

    if (now - lastDetectedAt < DETECT_COOLDOWN_MS) return;

    if (matchedNominal > 0) {
        lastDetectedAt = now;
        startConfirmation(matchedNominal);
    }
}

int matchNominal(int r, int g, int b, long &outDistance, int
&outBestNominal) {
    int bestMatch = -1;
    long bestDistance = 999999;
    outBestNominal = -1;

    for (int i = 0; i < 5; i++) {
        if (!calibrationData[i].valid) continue;

        long dr = r - calibrationData[i].r;
        long dg = g - calibrationData[i].g;
        long db = b - calibrationData[i].b;
        long distance = sqrt((double)(dr*dr + dg*dg + db*db));

        if (distance < bestDistance) {
```

```
            bestDistance = distance;
            outBestNominal = calibrationData[i].nominal;
            if (distance <= calibrationData[i].threshold) {
                bestMatch = calibrationData[i].nominal;
            }
        }
    }
    outDistance = bestDistance;
    return bestMatch;
}


// ==========================================
// KONFIRMASI NOMINAL — tekan 0=benar, 1=batal
// ==========================================
void startConfirmation(int nominal) {
    pendingNominal = nominal;
    confirmationMode = true;
    confirmationStartAt = millis();

    // FIX: buffer diperbesar dari 17 -> 32 supaya tidak overflow
(stack smashing)
    // "Rp 100000 terdeteksi" butuh 21 byte, buffer lama 17 byte
tidak cukup
    char line1[32];
    sprintf(line1, "Rp %d terdeteksi", nominal);
    lcdShowMessage(String(line1), "0=Benar 1=Batal");

    buzzerBeep(1, 80);
}

void handleConfirmationTimeout() {
    if (!confirmationMode) return;

    if (millis() - confirmationStartAt >= CONFIRMATION_TIMEOUT_MS) {
        Serial.println("[KONFIRMASI] Timeout, otomatis dibatalkan");
        cancelConfirmation();
    }
}

void confirmNominal() {
    Serial.print("[KONFIRMASI] Dikonfirmasi benar, kirim ke server:
Rp");
    Serial.println(pendingNominal);

    onMoneyConfirmed(pendingNominal);
```

```
    confirmationMode = false;
    pendingNominal = 0;
}

void cancelConfirmation() {
    Serial.println("[KONFIRMASI] Dibatalkan oleh user/timeout");

    lcdShowMessage("Dibatalkan", "");
    buzzerBeep(1, 300);
    delay(1000);

    confirmationMode = false;
    pendingNominal = 0;
    showIdleScreen();
}

void onMoneyConfirmed(int nominal) {
    JsonDocument doc;
    doc["device_id"] = DEVICE_ID;
    doc["nominal"] = nominal;
    publishJson(TOPIC_MONEY, doc);

    char buf[32];
    sprintf(buf, "Rp %d masuk", nominal);
    lcdShowMessage(buf, "Terkirim...");

    ledBlinkHijau();
    buzzerBeep(1, 100);

    delay(1500);
    showIdleScreen();
}


// ==========================================
// KEYPAD — input PIN, mode kalibrasi, mode baca, konfirmasi, lihat
saldo
// ==========================================
void handleKeypad() {
    char key = keypad.getKey();
    if (!key) return;

    Serial.print("[KEYPAD] Tombol: ");
    Serial.println(key);

    // Prioritas paling atas: kalau sedang mode konfirmasi nominal
    if (confirmationMode) {
```

```
        if (key == '0') {
            confirmNominal();
        } else if (key == '1') {
            cancelConfirmation();
        }
        return; // tombol lain diabaikan selama mode konfirmasi
aktif
    }

    // Tombol C = lihat total tabungan (tampil sementara, lalu balik
normal)
    if (key == 'C') {
        showSaldoScreen();
        return;
    }

    // Tombol B = masuk/keluar mode baca real-time
    if (key == 'B') {
        readingMode = !readingMode;
        if (readingMode) {
            lcdShowMessage("Mode Baca ON", "Tekan B keluar");
            delay(800);
        } else {
            showIdleScreen();
        }
        return;
    }

    // Tombol A = masuk mode kalibrasi
    if (key == 'A' && !calibrationMode) {
        calibrationMode = true;
        lcdShowMessage("Mode Kalibrasi", "Pilih 1-5");
        buzzerBeep(2, 100);
        return;
    }

    if (calibrationMode) {
        if (key >= '1' && key <= '5') {
            calibrationTargetIndex = key - '1';
            doCalibration(calibrationTargetIndex);
            calibrationMode = false;
            showIdleScreen();
        } else if (key == 'D') {
            calibrationMode = false;
            showIdleScreen();
        }
        return;
```

```
    }

    // Input PIN normal
    if (key == '#') {
        if (pinBuffer.length() > 0) {
            submitPin(pinBuffer);
            pinBuffer = "";
        }
    } else if (key == '*') {
        pinBuffer = "";
        lcdShowMessage("Masukkan PIN:", "");
    } else if (key >= '0' && key <= '9') {
        pinBuffer += key;
        String masked = "";
        for (unsigned int i = 0; i < pinBuffer.length(); i++) masked
+= "*";
        lcdShowMessage("Masukkan PIN:", masked);
    }
}

void doCalibration(int index) {
    uint16_t r, g, b, c;
    tcs.getRawData(&r, &g, &b, &c);

    if (c < 10) {
        lcdShowMessage("Kalibrasi Gagal", "Dekatkan objek");
        buzzerBeep(3, 100);
        return;
    }

    int rNorm = (int)((long)r * 1000 / c);
    int gNorm = (int)((long)g * 1000 / c);
    int bNorm = (int)((long)b * 1000 / c);

    JsonDocument doc;
    doc["device_id"] = DEVICE_ID;
    doc["nominal"] = nominalList[index];
    doc["r"] = rNorm;
    doc["g"] = gNorm;
    doc["b"] = bNorm;
    doc["threshold"] = 40;
    publishJson(TOPIC_CALIBRATION, doc);

    Serial.print("[KALIBRASI] Nominal ");
    Serial.print(nominalList[index]);
    Serial.print(" -> Rnorm:"); Serial.print(rNorm);
    Serial.print(" Gnorm:"); Serial.print(gNorm);
```

```
    Serial.print(" Bnorm:"); Serial.println(bNorm);
    Serial.println("[KALIBRASI] Menunggu server balas konfirmasi via
TOPIC_CONFIG...");

    lcdShowMessage("Kalibrasi", "Terkirim!");
    buzzerBeep(1, 150);
    delay(1000);
}

void submitPin(String pin) {
    JsonDocument doc;
    doc["device_id"] = DEVICE_ID;
    doc["pin"] = pin;
    publishJson(TOPIC_PIN, doc);

    lcdShowMessage("Memeriksa PIN...", "");
}


// ==========================================
// RELAY (Solenoid Lock)
// ==========================================
void relayWrite(bool active) {
    if (RELAY_ACTIVE_HIGH) {
        digitalWrite(RELAY_PIN, active ? HIGH : LOW);
    } else {
        digitalWrite(RELAY_PIN, active ? LOW : HIGH);
    }
}

void activateRelay() {
    relayWrite(true);
    relayActive = true;
    relayActivatedAt = millis();
    publishActivity("servo_buka", "Solenoid aktif");
}

void handleRelayTimer() {
    if (relayActive && millis() - relayActivatedAt >=
RELAY_DURATION_MS) {
        relayWrite(false);
        relayActive = false;
        publishActivity("servo_tutup", "Solenoid nonaktif");
    }
}
```

```
// ==========================================
// LED & BUZZER
// ==========================================
void ledBlinkHijau() {
    digitalWrite(LED_HIJAU_PIN, HIGH);
    delay(200);
    digitalWrite(LED_HIJAU_PIN, LOW);
}

void ledBlinkMerah() {
    digitalWrite(LED_MERAH_PIN, HIGH);
    delay(200);
    digitalWrite(LED_MERAH_PIN, LOW);
}

void buzzerBeep(int times, int durationMs) {
    for (int i = 0; i < times; i++) {
        digitalWrite(BUZZER_PIN, HIGH);
        delay(durationMs);
        digitalWrite(BUZZER_PIN, LOW);
        delay(100);
    }
}


// ==========================================
// LCD HELPER
// ==========================================
void lcdShowMessage(String line1, String line2) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print(line1);
    lcd.setCursor(0, 1);
    lcd.print(line2);
}

void showIdleScreen() {
    lcdShowMessage("Celengan Siap", "");
}

void showSaldoScreen() {
    if (cachedTotalTabungan >= 0) {
        char line2[17];
        sprintf(line2, "Rp%ld", cachedTotalTabungan);
        lcdShowMessage("Total Tabungan:", String(line2));
    } else {
        lcdShowMessage("Total Tabungan:", "Belum ada data");
```

```
    }
    delay(2000);
    showIdleScreen();
}
```