

Capstone Project: Guardian-Track Access System

1. The Problem: School Security Vulnerabilities

Management of student movement in large Kenyan boarding institutions remains a massive logistical challenge. Currently, schools rely on manual paper registers and physical security guards. This creates three critical vulnerabilities:

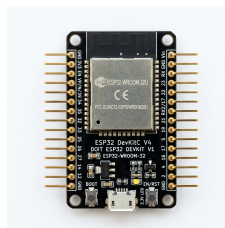
1. Proxy Attendance: Students signing in for absent friends in dormitories or classrooms.
2. Unauthorized Exits ('Sneaking Out'): Exploiting guard fatigue or forging paper gate passes.
3. Information Lag: Principals have no real-time dashboard to know the exact location of the student body during an emergency.

2. The Solution

Guardian-Track is a scalable, low-cost Embedded IoT Security Solution tailored for educational institutions. It replaces unreliable paper logs with an automated RFID 'Choke Point' framework. Unlike expensive biometric systems that require heavy processing power and raise privacy concerns, this embedded system utilizes lightweight 'tap-and-go' RFID technology. By leveraging low-cost microcontrollers (ESP32) and decentralized nodes, the system offers high reliability, low power consumption, and real-time data synchronization. The system guarantees a highly accurate digital roll call and instantaneously actuates mechanical barriers and alarms upon unauthorized exit attempts.

3. Bill of Materials (Prototype Requirements)

- 1x ESP32 Development Board



- 3x MFRC522 RFID Reader Modules



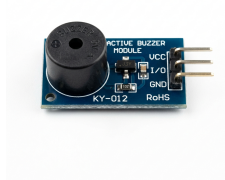
- 1x I2C OLED Display (0.96")



- 1x SG90 Micro Servo Motor



- 1x Active Buzzer Module



- 2x LED Status Indicators (Red & Green)



4. Build Procedure

1. Bus Initialization: Wire 3 MFRC522 readers to the shared ESP32 SPI pins (SCK:18, MISO:19, MOSI:23). Assign distinct Slave Select pins (GPIO 5, 25, 26). Connect the I2C OLED to SDA (21) and SCL (22).
2. Actuator & Indicator Integration: Connect the SG90 Servo to GPIO 4, Active Buzzer to GPIO 2, Red LED to GPIO 32, and Green LED to GPIO 33.
3. Code Flash: Compile and flash 'guardian_track_pro.ino' to the ESP32. Ensure the ESP32 is powered via a stable 5V source to handle the servo load.
4. Network Configuration: The ESP32 will automatically broadcast an Access Point named 'GuardianTrack-Hub'. Connect to this network to access the live dashboard.

5. Expected Outcome

Upon booting, the ESP32 initializes the RFIDs and displays system status on the OLED. When a student scans their card, the OLED immediately reflects their name and location, while the Green LED blinks. If an authorized student scans at the Gate, the Servo motor actuates 90 degrees to open the barrier. If an unauthorized card is scanned at the Gate, or if a 'double-exit' is attempted (Anti-Passback violation), the barrier remains locked, the Red LED flashes, the Buzzer sounds a loud alarm, and the OLED displays an Alert.

6. Technical Architecture & Syllabus Application

This Capstone is heavily rooted in Embedded Systems principles, serving as a direct application of the Gearbox Academy Track A Syllabus. Key embedded concepts utilized include:

- SPI Communication (Week 9): Interfacing three separate MFRC522 readers simultaneously on a shared SPI bus using distinct Chip Select (CS) pins.
- I2C Displays (Week 10): Integrating a 0.96" SSD1306 OLED via I2C (SDA/SCL) to provide real-time visual feedback directly on the hardware hub.
- IoT & Web Servers (Week 10): Leveraging the ESP32 to host an Access Point (AP) and serve dynamic HTML dashboards over Wi-Fi without internet reliance.
- Actuator & Relay Logic (Week 11): Translating low-voltage GPIO signals into mechanical action (Servo) and auditory/visual alarms (Buzzer & LEDs).
- Embedded C++: Memory management of UID strings and structured state machines for Anti-Passback logic.

7. Source Code - Prototype Implementation

```
// Guardian-Track Commercial Edition: Central Hub
// Sub-system: ESP32 IoT Access Control Dashboard
// Author: Kassim Musa Abass

#include <SPI.h>
#include <MFRC522.h>
#include <WiFi.h>
#include <WebServer.h>
#include <ESP32Servo.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// -----
// PIN DEFINITIONS (ESP32)
// -----
// Shared SPI Pins
#define SCK_PIN 18
#define MISO_PIN 19
#define MOSI_PIN 23
#define RST_PIN 15 // Shared Reset for all MFRC522

// Unique Slave Select (CS) Pins for the 3 Zones
#define SS_GATE 5
#define SS_CLASS 25
#define SS_DORM 26

// Actuators & Indicators
#define SERVO_PIN 4
#define BUZZER_PIN 2
#define RED_LED 32
#define GREEN_LED 33

// I2C OLED Pins (SDA = 21, SCL = 22 by default)
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// -----
// PERIPHERAL INSTANCES
// -----
MFRC522 rfidGate(SS_GATE, RST_PIN);
MFRC522 rfidClass(SS_CLASS, RST_PIN);
MFRC522 rfidDorm(SS_DORM, RST_PIN);

Servo gateBarrier;
WebServer server(80);

// -----
// SYSTEM STATE & LOGS
```

```

// -----
struct Student {
    String uid;
    String name;
    String currentZone;
    bool authorizedToLeave;
};

// Mock Database of Students
Student database[2] = {
    {"04 EA 34 52", "Kassim Musa", "Dormitory", false},
    {"A3 4B 9C 11", "John Doe", "Classroom", true} // John has a valid Gate Pass
};

// Activity Log
String activityLog = "";

// -----
// WIFI CONFIG (Access Point Mode)
// -----
const char* apSSID = "GuardianTrack-Hub";
const char* apPass = "admin1234";

// -----
// WEB DASHBOARD HTML
// -----
void handleRoot() {
    String html = "<html><head><title>Guardian-Track Dashboard</title>";
    html += "<meta name='viewport' content='width=device-width, initial-scale=1.0'>";
    html += "<style>";
    html += "body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; background-color: #f4f7f6; color: #333; margin: 0; padding: 20px; }";
    html += "h1 { color: #2c3e50; text-align: center; }";
    html += ".container { max-width: 800px; margin: auto; background: white; padding: 20px; border-radius: 10px; box-shadow: 0 4px 8px rgba(0,0,0,0.1); }";
    html += ".log-box { background: #1e272e; color: #0be881; padding: 15px; border-radius: 5px; height: 300px; overflow-y: auto; font-family: monospace; }";
    html += "</style></head><body>";

    html += "<div class='container'>";
    html += "  <h1>Guardian-Track Central Dashboard</h1>";
    html += "  <p><b>System Status:</b> Online | <b>Active Nodes:</b> 3 (Gate, Class, Dorm)</p>";

    html += "  <h3>Live Activity Log</h3>";
    html += "  <div class='log-box'>" + activityLog + "</div>";
    html += "  <p style='text-align:center; margin-top:20px;'><button onclick='location.reload()' style='padding: 10px 20px; background: #0984e3; color: white; border: none; border-radius: 5px; cursor: pointer;'>Refresh Dashboard</button></p>";
    html += "</div>";

    html += "</body></html>";
    server.send(200, "text/html", html);
}

// -----
// DISPLAY HELPER
// -----
void updateDisplay(String line1, String line2) {
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.println("Guardian-Track Hub");
    display.drawLine(0, 10, 128, 10, WHITE);

    display.setCursor(0, 20);
    display.println(line1);
    display.setCursor(0, 40);
    display.println(line2);
    display.display();
}

// -----
// SETUP
// -----

```

```

void setup() {
    Serial.begin(115200);

    // Display Setup
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println(F("SSD1306 allocation failed"));
    }
    updateDisplay("System Booting...", "Initializing SPI");

    // Actuators & Indicators Setup
    gateBarrier.attach(SERVO_PIN);
    gateBarrier.write(0); // Gate Closed
    pinMode(BUZZER_PIN, OUTPUT);
    pinMode(RED_LED, OUTPUT);
    pinMode(GREEN_LED, OUTPUT);
    digitalWrite(BUZZER_PIN, LOW);
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, LOW);

    // SPI Setup
    SPI.begin(SCK_PIN, MISO_PIN, MOSI_PIN, -1);

    // Initialize RFIDs
    updateDisplay("Initializing RFIDs", "Gate, Class, Dorm");
    rfidGate.PCD_Init();
    rfidClass.PCD_Init();
    rfidDorm.PCD_Init();

    // Network Setup (Host AP)
    updateDisplay("Starting AP...", apSSID);
    WiFi.softAP(apSSID, apPass);

    // Web Server Setup
    server.on("/", handleRoot);
    server.begin();

    logActivity("System Initialized.");
    updateDisplay("System Ready!", "Awaiting Scans...");
}

// -----
// MAIN LOOP
// -----
void loop() {
    server.handleClient();

    // Check Dormitory Node
    if (rfidDorm.PICC_IsNewCardPresent() && rfidDorm.PICC_ReadCardSerial()) {
        processScan(&rfidDorm, "Dormitory");
    }

    // Check Classroom Node
    if (rfidClass.PICC_IsNewCardPresent() && rfidClass.PICC_ReadCardSerial()) {
        processScan(&rfidClass, "Classroom");
    }

    // Check Main Gate Node (Choke Point)
    if (rfidGate.PICC_IsNewCardPresent() && rfidGate.PICC_ReadCardSerial()) {
        processGateScan(&rfidGate);
    }
}

// -----
// LOGIC HANDLERS
// -----
String getUIDString(MFRC522* rfid) {
    String uid = "";
    for (byte i = 0; i < rfid->uid.size; i++) {
        uid += String(rfid->uid.uidByte[i] < 0x10 ? " 0" : " ");
        uid += String(rfid->uid.uidByte[i], HEX);
    }
    uid.trim();
    uid.toUpperCase();
    return uid;
}

```

```

Student* findStudent(String uid) {
    for (int i = 0; i < 2; i++) {
        if (database[i].uid == uid) return &database[i];
    }
    return nullptr;
}

void processScan(MFRC522* rfid, String zone) {
    String uid = getUIDString(rfid);
    Student* student = findStudent(uid);

    if (student != nullptr) {
        student->currentZone = zone;
        logActivity(student->name + " in " + zone);
        updateDisplay(student->name, "Entered " + zone);

        // Quick Green Blink
        digitalWrite(GREEN_LED, HIGH);
        delay(500);
        digitalWrite(GREEN_LED, LOW);
    } else {
        logActivity("Unknown Card at " + zone);
        updateDisplay("Unknown Card", zone);

        // Quick Red Blink
        digitalWrite(RED_LED, HIGH);
        delay(500);
        digitalWrite(RED_LED, LOW);
    }
    rfid->PICC_HaltA();
    delay(1500);
    updateDisplay("System Ready!", "Awaiting Scans...");
}

void processGateScan(MFRC522* rfid) {
    String uid = getUIDString(rfid);
    Student* student = findStudent(uid);

    if (student == nullptr) {
        logActivity("SECURITY ALERT: Unknown Card at Gate!");
        updateDisplay("ALERT: Unknown", "Access Denied");
        triggerAlarm();
    } else if (!student->authorizedToLeave) {
        logActivity("SECURITY ALERT: " + student->name + " attempted sneak out!");
        updateDisplay("ALERT: " + student->name, "No Gate Pass!");
        triggerAlarm();
    } else if (student->currentZone == "Outside") {
        // Anti-Passback logic
        logActivity("SECURITY ALERT: " + student->name + " Anti-Passback Violation!");
        updateDisplay("ALERT: Passback", student->name);
        triggerAlarm();
    } else {
        // Valid Gate Pass
        logActivity("Access Granted: " + student->name + " exited.");
        updateDisplay("Access Granted", student->name + " Exiting");
        student->currentZone = "Outside";
        openGate();
    }
    rfid->PICC_HaltA();
    updateDisplay("System Ready!", "Awaiting Scans...");
}

void logActivity(String msg) {
    String timestamp = String(millis() / 1000) + "s : ";
    activityLog = timestamp + msg + "<br>" + activityLog;
    Serial.println(msg);
}

void openGate() {
    digitalWrite(GREEN_LED, HIGH);
    gateBarrier.write(90); // Open
    delay(3000);
    gateBarrier.write(0); // Close
    digitalWrite(GREEN_LED, LOW);
}

```

```
}  
  
void triggerAlarm() {  
    for (int i = 0; i < 5; i++) {  
        digitalWrite(RED_LED, HIGH);  
        digitalWrite(BUZZER_PIN, HIGH);  
        delay(200);  
        digitalWrite(RED_LED, LOW);  
        digitalWrite(BUZZER_PIN, LOW);  
        delay(200);  
    }  
}
```