```
// DVE R3 (Arduino Due) Code - The Sensor Hub
// FINAL VERSION - CONFIGURED FOR DHT11

#include "DHT.h"
#include <Servo.h>              // Include the Servo library
#include <Wire.h>               // I2C Communication Library
#include <LiquidCrystal_I2C.h>  // LCD Library
#include <ArduinoJson.h>

// --- Pin Definitions ---
#define DHTPIN 3
#define DHTTYPE DHT11
#define SOIL_PIN A0
#define LDR_PIN1 A1  // Pin for the LDR voltage divider
#define LDR_PIN2 A2
#define LDR_PIN3 A3
#define SERVO_PIN1 9  // PWM Pin for the servo signal
#define SERVO_PIN2 10
#define SERVO_PIN3 11
#define Water_Level_Pin 4
#define Nutrient_Level_Pin 5
#define GROW_LIGHT_PIN 7
#define FAN_PIN 8
#define WATER_PUMP_PIN 12
#define NUTRIENT_PUMP_PIN 13

LiquidCrystal_I2C lcd(0x27, 16, 2);
```

```
// Layer 3
const int lightThresholdOpen_default[3] = { 750, 750, 750 };
const int fanOnHumidity_default = 75;
const int fanOnTemperature_default = 29;
const int wateringThreshold_default = 900;
const long wateringDurationMs_default = 1000;
const int lowHumidityWatering_default = 40;
const int lightOnHour_default = 8;
const int lightOffHour_default = 22;

int targetAngle[3] = { 60, 60, 60 };
int currentAngle[3] = { 60, 60, 60 };
unsigned long lastServoMoveTime = 0;
const int servoStepInterval = 100;  // Time in ms between each 1-degree step. HIGHER = SLOWER.

// Time-keeping variable, updated by ESP8266
int currentHour = -1;   // -1 means time is not yet synced
int previousHour = -1;  // To detect when the hour changes

// Flag to ensure nutrient pump runs only once per scheduled hour
bool nutrientPumpedThisHour = false;

// Nutrient Pump Scheduling (Layer 3 - Defaults)
const int nutrientOnHour_default = 10;         // Default: 10 AM
const long nutrientDurationMs_default = 1500;  // Default: 1.5 seconds
```

```
// Nutrient Pump Scheduling (Layer 2 - Active)
int nutrientOnHour_active = nutrientOnHour_default;
long nutrientDurationMs_active = nutrientDurationMs_default;

// Layer 2
int lightThresholdOpen_active[3];
int fanOnHumidity_active = fanOnHumidity_default;
int fanOnTemperature_active = fanOnTemperature_default;
int wateringThreshold_active = wateringThreshold_default;
long wateringDurationMs_active = wateringDurationMs_default;
int lowHumidityWatering_active = lowHumidityWatering_default;
int lightOnHour_active = lightOnHour_default;
int lightOffHour_active = lightOffHour_default;

// DHT Sensor
DHT dht(DHTPIN, DHTTYPE);
Servo coverServo1;
Servo coverServo2;
Servo coverServo3;

// Variables for the non-blocking automation timer
unsigned long previousAutomationMillis = 0;
// Set the interval for how often to run the check
const long automationInterval = 4000;

void setup() {
```

```
Serial.begin(9600);
Serial1.begin(9600);  // Serial1 is for communication with the ESP8266

pinMode(19, INPUT_PULLUP);  // The critical noise-prevention fix

pinMode(Water_Level_Pin, INPUT);
pinMode(Nutrient_Level_Pin, INPUT);
pinMode(GROW_LIGHT_PIN, OUTPUT);
pinMode(FAN_PIN, OUTPUT);
pinMode(WATER_PUMP_PIN, OUTPUT);
pinMode(NUTRIENT_PUMP_PIN, OUTPUT);

digitalWrite(GROW_LIGHT_PIN, LOW);
digitalWrite(FAN_PIN, LOW);
digitalWrite(WATER_PUMP_PIN, HIGH);
digitalWrite(NUTRIENT_PUMP_PIN, HIGH);  // HIGH = OFF

revertToDefaults();

// Start the DHT sensor
dht.begin();

Wire.begin();
lcd.init();
lcd.backlight();
lcd.setCursor(0, 0);
```

```
  lcd.print("PodPal Starting...");

  // --- ADD THIS BLOCK TO FIX THE PROBLEM ---
  coverServo1.attach(SERVO_PIN1);
  coverServo2.attach(SERVO_PIN2);
  coverServo3.attach(SERVO_PIN3);

  // Set a neutral starting position for all servos
  coverServo1.write(60);
  coverServo2.write(60);
  coverServo3.write(60);
  // ----------------------------------------

  Serial.println("Arduino setup complete. Autonomous logic is active.");
}

void executeRealTimeLogic() {
  int ldr1 = analogRead(LDR_PIN1);
  int ldr2 = analogRead(LDR_PIN2);
  int ldr3 = analogRead(LDR_PIN3);
  float temp = dht.readTemperature();
  float humidity = dht.readHumidity();
  int moisture = analogRead(SOIL_PIN);

  // int currentHour = getCurrentHour(); // Placeholder for time logic

  // --- Only run time-based logic if time has been synced ---
```

```
if (currentHour != -1) {

  // --- Check if the hour has changed to reset single-shot tasks ---
  if (currentHour != previousHour) {
    nutrientPumpedThisHour = false;  // Reset the flag for the new hour
    previousHour = currentHour;      // Update the previous hour
  }

  // --- Instinct: Scheduled Nutrient Pump ---
  if (currentHour == nutrientOnHour_active && !nutrientPumpedThisHour) {
    Serial.println("Instinct: Scheduled nutrient cycle.");
    digitalWrite(NUTRIENT_PUMP_PIN, HIGH);
    delay(nutrientDurationMs_active);
    digitalWrite(NUTRIENT_PUMP_PIN, LOW);
    nutrientPumpedThisHour = true;  // Set flag to prevent re-running this hour
  }
}

// --- Set Servo Target Angles based on LDR values ---
// Servo 1 Target
if (ldr1 > lightThresholdOpen_active[0]) {
  targetAngle[0] = 90;
} else {
  targetAngle[0] = 60;
}

// Servo 2 Target
```

```
if (ldr2 > lightThresholdOpen_active[1]) {
  targetAngle[1] = 90;
} else {
  targetAngle[1] = 60;
}

// Servo 3 Targets
if (ldr3 > lightThresholdOpen_active[2]) {
  targetAngle[2] = 90;
} else {
  targetAngle[2] = 60;
}

if (!isnan(temp) && temp > fanOnTemperature_active) {
  digitalWrite(FAN_PIN, LOW);
} else {
  digitalWrite(FAN_PIN, HIGH);
}

// CORRECTED watering logic
if (moisture > wateringThreshold_active) {
  // If the soil is dry...
  Serial.println("Instinct: Soil dry. Main watering cycle.");
  digitalWrite(WATER_PUMP_PIN, LOW);  // Send LOW to turn the pump ON
  delay(wateringDurationMs_active);
  digitalWrite(WATER_PUMP_PIN, HIGH);  // Send HIGH to turn the pump OFF
} else {
```

```
  // Otherwise, ensure the pump is OFF
  digitalWrite(WATER_PUMP_PIN, HIGH);  // Send HIGH to keep the pump OFF
}

if (currentHour >= lightOnHour_active && currentHour < lightOffHour_active) {
  digitalWrite(GROW_LIGHT_PIN, LOW);
} else {
  digitalWrite(GROW_LIGHT_PIN, HIGH);
}

// This will print the complete status of all sensors and actuators.
Serial.println("---------- PodPal Real-Time Status ----------");

// LDR and Servo Status
Serial.print("LDR 1: ");
Serial.print(ldr1);
Serial.print(" \t| Servo 1 Angle: ");
Serial.println(coverServo1.read());

Serial.print("LDR 2: ");
Serial.print(ldr2);
Serial.print(" \t| Servo 2 Angle: ");
Serial.println(coverServo2.read());

Serial.print("LDR 3: ");
Serial.print(ldr3);
Serial.print(" \t| Servo 3 Angle: ");
```

```
  Serial.println(coverServo3.read());
  Serial.println();  // Blank line

  // Actuator Status
  Serial.print("Fan Status: ");
  Serial.println(digitalRead(FAN_PIN) == LOW ? "ON" : "OFF");

  Serial.print("Grow Light: ");
  Serial.println(digitalRead(GROW_LIGHT_PIN) == LOW ? "ON" : "OFF");

  Serial.println("-----------------------------------------");
  Serial.println();  // Extra blank line for readability
  // -----------------------------------------------
}

/**
 * Parses the new AI plan from the ESP8266 and updates ALL active thresholds.
 */
void parseAndUpdatePlan(String planJson) {
  Serial.println("!!! AI Plan Received! Calling parseAndUpdatePlan()...");
  StaticJsonDocument<1024> doc;  // Increased size for all keys
  DeserializationError error = deserializeJson(doc, planJson);
  if (error) {
    Serial.print("JSON Error: ");
    Serial.println(error.c_str());
    return;
  }
```

```
  Serial.println("New AI Plan Received. Updating all thresholds.");

  JsonArray open_thresholds = doc["light_threshold_open"];

  // Check if the arrays were sent and have the correct size (3)
  if (!open_thresholds.isNull() && open_thresholds.size() == 3) {
    for (int i = 0; i < 3; i++) {
      lightThresholdOpen_active[i] = open_thresholds[i] | lightThresholdOpen_default[i];
    }
  }

  fanOnHumidity_active = doc["fan_on_humidity"] | fanOnHumidity_default;
  fanOnTemperature_active = doc["fan_on_temperature"] | fanOnTemperature_default;
  wateringThreshold_active = doc["watering_threshold"] | wateringThreshold_default;
  wateringDurationMs_active = doc["watering_duration_ms"] |
wateringDurationMs_default;
  lowHumidityWatering_active = doc["low_humidity_watering_threshold"] |
lowHumidityWatering_default;
  lightOnHour_active = doc["light_on_hour"] | lightOnHour_default;
  lightOffHour_active = doc["light_off_hour"] | lightOffHour_default;

  nutrientOnHour_active = doc["nutrient_on_hour"] | nutrientOnHour_default;
  nutrientDurationMs_active = doc["nutrient_duration_ms"] | nutrientDurationMs_default;

  Serial.print("--> Plan Parsed. New active close threshold is: ");
  Serial.println(lightThresholdOpen_active[0]);  // Print one of the new values as proof
```

```
  Serial1.println("PLAN_OK");
}

/**
 * Reverts ALL active thresholds to their failsafe defaults. This is Layer 3.
 */
void revertToDefaults() {
  for (int i = 0; i < 3; i++) {
    lightThresholdOpen_active[i] = lightThresholdOpen_default[i];
  }

  fanOnHumidity_active = fanOnHumidity_default;
  fanOnTemperature_active = fanOnTemperature_default;
  wateringThreshold_active = wateringThreshold_default;
  wateringDurationMs_active = wateringDurationMs_default;
  lowHumidityWatering_active = lowHumidityWatering_default;
  lightOnHour_active = lightOnHour_default;
  lightOffHour_active = lightOffHour_default;
  nutrientOnHour_active = nutrientOnHour_default;
  nutrientDurationMs_active = nutrientDurationMs_default;
  Serial.println("FAILSAFE: Reverted to default brain.");
}

void updateLcdDisplay() {
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();
```

```
// A LOW reading means the liquid is DETECTED (level is OK)
bool waterLevelOk = (digitalRead(Water_Level_Pin) == LOW);
bool nutrientLevelOk = (digitalRead(Nutrient_Level_Pin) == LOW);

// Check failed
if (isnan(temp) || isnan(hum)) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Sensor Error");
  return;
}

// Clear the current values and print new Data
lcd.clear();

lcd.setCursor(0, 0);
lcd.print("T:");
lcd.print(temp, 1);
lcd.print(" C` H:");
lcd.print(hum, 0);
lcd.print("%");

lcd.setCursor(0, 1);
lcd.print("W:");
lcd.print(waterLevelOk ? "OK " : "LOW ");  // Use ternary operator for clean code

lcd.setCursor(8, 1);
```

```
  lcd.print("N:");
  lcd.print(nutrientLevelOk ? "OK" : "LOW");
}

void updateServos() {
  // Check if it's time to move the servos one step
  if (millis() - lastServoMoveTime >= servoStepInterval) {
    lastServoMoveTime = millis();  // Update the timer

    // Loop through all three servos
    for (int i = 0; i < 3; i++) {
      // Check if the servo needs to move
      if (currentAngle[i] != targetAngle[i]) {

        // Move one step closer to the target
        if (currentAngle[i] < targetAngle[i]) {
          currentAngle[i]++;
        } else {
          currentAngle[i]--;
        }

        // Send the new position to the correct servo
        switch (i) {
          case 0:
            coverServo1.write(currentAngle[i]);
            break;
          case 1:
```

```
          coverServo2.write(currentAngle[i]);
          break;
        case 2:
          coverServo3.write(currentAngle[i]);
          break;
      }
    }
  }
}

void loop() {
  // === TASK 1: Handle Automation ===
  // This non-blocking timer checks if it's time to run the automation.
  unsigned long currentMillis = millis();
  if (currentMillis - previousAutomationMillis >= automationInterval) {
    // Save the last time we ran the logic
    previousAutomationMillis = currentMillis;

    // Single call to the new, comprehensive logic function
    executeRealTimeLogic();

    updateLcdDisplay();  // LCD function
  }

  // This function runs constantly to handle smooth servo movement
  updateServos();
```

```
//Task 2
// Check if a command has been sent from the ESP8266
if (Serial1.available() > 0) {
  String command = Serial1.readStringUntil('\n');
  command.trim();

  if (command.startsWith("plan:")) {
    String jsonPayload = command.substring(5);
    parseAndUpdatePlan(jsonPayload);
  }
  // --- To handle the time command ---
  else if (command.startsWith("time:")) {
    String hourString = command.substring(5);
    currentHour = hourString.toInt();
    Serial.print("Time Updated. Current Hour is now: ");
    Serial.println(currentHour);
  }

  else if (command == "USE_DEFAULTS") {
    revertToDefaults();  // Use the new function name
  }

  // --- Respond to specific data requests ---
  else if (command == "get_temp") {
    float temperature = dht.readTemperature();
    if (isnan(temperature)) {
```

```
    Serial1.println("err");
  } else {
    Serial1.println(temperature);
  }
} else if (command == "get_humidity") {
  float humidity = dht.readHumidity();
  if (isnan(humidity)) {
    Serial1.println("err");
  } else {
    Serial1.println(humidity);
  }
} else if (command == "get_moisture") {
  int moistureValue = analogRead(SOIL_PIN);
  Serial1.println(moistureValue);
}
// --- NEW: LDR/Servo Commands ---
else if (command == "get_ldr1") {
  int ldrValue1 = analogRead(LDR_PIN1);
  Serial1.println(ldrValue1);
} else if (command == "get_ldr2") {
  int ldrValue2 = analogRead(LDR_PIN2);
  Serial1.println(ldrValue2);
} else if (command == "get_ldr3") {
  int ldrValue3 = analogRead(LDR_PIN3);
  Serial1.println(ldrValue3);
}
```

```
    else if (command == "get_servo_angle1") {
      int angle1 = coverServo1.read();
      Serial1.println(angle1);
    } else if (command == "get_servo_angle2") {
      int angle2 = coverServo2.read();
      Serial1.println(angle2);
    } else if (command == "get_servo_angle3") {
      int angle3 = coverServo3.read();
      Serial1.println(angle3);
    }

    else if (command == "get_water_level") {
      bool waterLevelOk = (digitalRead(Water_Level_Pin) == LOW);
      Serial1.println(waterLevelOk ? "OK" : "LOW");
    } else if (command == "get_nutrient_level") {
      bool nutrientLevelOk = (digitalRead(Nutrient_Level_Pin) == LOW);
      Serial1.println(nutrientLevelOk ? "OK" : "LOW");
    }
  }
}
```