Demeter - Hardware Engineer PRD

ESP32 Sensor Node & Synthetic Data

Role: Hardware/IoT Engineer
Stack: ESP32 + Arduino + DHT22 + Capacitive Soil Sensor

---

1. Overview

Build the IoT sensor node that collects soil moisture, temperature, and humidity data from the farm and transmits it to the cloud API. Also build a synthetic data generator as fallback for demos.

---

2. Hardware Components

Component	Model	Purpose	Cost
Microcontroller	ESP32 DevKit V1	WiFi + Processing	~$5
Temp/Humidity	DHT22 (AM2302)	Air temperature & humidity	~$3
Soil Moisture	Capacitive v1.2	Soil water content	~$2
Power	5V USB / 18650 + Solar	Continuous operation	~$5-15
Enclosure	IP65 Junction Box	Weather protection	~$5

Total BOM: ~$20-30 per node

---

3. Wiring Diagram

```
                    +-------------------+
                    |    ESP32 DevKit   |
                    |                   |
    DHT22           |                   |           Soil Moisture
   +-------+        |                   |            +---------+
   | VCC  |---------|  3.3V             |            |  VCC   |-------- 3.3V
   | DATA |---------|  GPIO4            |            |  GND   |-------- GND
   | NC   |         |                   |            |  AOUT  |-------- GPIO32
   | GND  |---------|  GND              |            +---------+
   +-------+        |                   |
                    |                   |
                    |  USB ------------|-----> Power (5V)
                    |                   |
                    |  WiFi Antenna    |-----> Cloud API
                    +-------------------+


Pin Assignments:
+----------+--------+------------------+
| Function | GPIO   | Notes            |
+----------+--------+------------------+
| DHT22    | GPIO4  | Digital input    |
| Soil     | GPIO32 | ADC1 Channel 4   |
| LED      | GPIO2  | Built-in LED     |
+----------+--------+------------------+
```

---

4. ESP32 Firmware

```cpp
// demeter_sensor.ino

#include <WiFi.h>
#include <HTTPClient.h>
#include <DHT.h>
#include <ArduinoJson.h>

// ============ CONFIGURATION ============
const char* WIFI_SSID = "YOUR_WIFI_SSID";
const char* WIFI_PASSWORD = "YOUR_WIFI_PASSWORD";
const char* API_ENDPOINT = "https://your-api.railway.app/api/v1/sensor-data";
const char* FARM_ID = "your-farm-uuid";

// Pin definitions
#define DHT_PIN 4
#define SOIL_PIN 32
#define LED_PIN 2
#define DHT_TYPE DHT22

// Reading interval (milliseconds)
#define READ_INTERVAL 300000  // 5 minutes

// Soil moisture calibration (adjust based on your sensor)
#define SOIL_DRY 3500    // ADC value when dry
#define SOIL_WET 1500    // ADC value when wet

// ============ GLOBALS ============
DHT dht(DHT_PIN, DHT_TYPE);
unsigned long lastReadTime = 0;

void setup() {
  Serial.begin(115200);
  delay(1000);

  Serial.println("\n=== DEMETER SENSOR NODE ===");

  // Initialize pins
  pinMode(LED_PIN, OUTPUT);
  pinMode(SOIL_PIN, INPUT);

  // Initialize DHT sensor
  dht.begin();

  // Connect to WiFi
  connectWiFi();

  // Initial reading
  sendSensorData();
}

void loop() {
  // Check WiFi connection
  if (WiFi.status() != WL_CONNECTED) {
    connectWiFi();
  }
```

```
  // Send data at interval
  if (millis() - lastReadTime >= READ_INTERVAL) {
    sendSensorData();
    lastReadTime = millis();
  }

  delay(1000);
}

void connectWiFi() {
  Serial.print("Connecting to WiFi");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 30) {
    delay(500);
    Serial.print(".");
    blinkLED(1, 100);
    attempts++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nWiFi connected!");
    Serial.print("IP: ");
    Serial.println(WiFi.localIP());
    blinkLED(3, 200);
  } else {
    Serial.println("\nWiFi connection failed!");
    blinkLED(5, 100);
  }
}

float readSoilMoisture() {
  // Take multiple readings and average
  int total = 0;
  for (int i = 0; i < 10; i++) {
    total += analogRead(SOIL_PIN);
    delay(10);
  }
  int avgReading = total / 10;

  // Convert to percentage (0-100%)
  float moisture = map(avgReading, SOIL_DRY, SOIL_WET, 0, 100);
  moisture = constrain(moisture, 0, 100);

  Serial.print("Soil ADC: ");
  Serial.print(avgReading);
  Serial.print(" -> ");
  Serial.print(moisture);
  Serial.println("%");

  return moisture;
}

void sendSensorData() {
  Serial.println("\n--- Reading Sensors ---");
```

```
// Read DHT22
float humidity = dht.readHumidity();
float temperature = dht.readTemperature();

// Check for read errors
if (isnan(humidity) || isnan(temperature)) {
  Serial.println("DHT read failed!");
  blinkLED(2, 500);
  return;
}

// Read soil moisture
float soilMoisture = readSoilMoisture();

// Print readings
Serial.print("Temperature: ");
Serial.print(temperature);
Serial.println(" C");
Serial.print("Humidity: ");
Serial.print(humidity);
Serial.println(" %");
Serial.print("Soil Moisture: ");
Serial.print(soilMoisture);
Serial.println(" %");

// Build JSON payload
StaticJsonDocument<256> doc;
doc["farm_id"] = FARM_ID;
doc["soil_moisture"] = round(soilMoisture * 10) / 10.0;
doc["temperature"] = round(temperature * 10) / 10.0;
doc["humidity"] = round(humidity * 10) / 10.0;
doc["timestamp"] = getISOTimestamp();

String jsonPayload;
serializeJson(doc, jsonPayload);

Serial.print("Payload: ");
Serial.println(jsonPayload);

// Send HTTP POST
if (WiFi.status() == WL_CONNECTED) {
  HTTPClient http;
  http.begin(API_ENDPOINT);
  http.addHeader("Content-Type", "application/json");

  int httpCode = http.POST(jsonPayload);

  if (httpCode > 0) {
    Serial.print("HTTP Response: ");
    Serial.println(httpCode);

    if (httpCode == 200 || httpCode == 201) {
      Serial.println("Data sent successfully!");
      blinkLED(1, 500);
    } else {
      Serial.print("Server error: ");
```

```
        Serial.println(http.getString());
        blinkLED(3, 200);
      }
    } else {
      Serial.print("HTTP Error: ");
      Serial.println(http.errorToString(httpCode));
      blinkLED(5, 100);
    }

    http.end();
  }
}

String getISOTimestamp() {
  // For accurate time, use NTP. Simplified version:
  // In production, sync with NTP server
  unsigned long ms = millis();
  char timestamp[25];
  sprintf(timestamp, "2026-02-25T%02d:%02d:%02dZ",
          (ms / 3600000) % 24,
          (ms / 60000) % 60,
          (ms / 1000) % 60);
  return String(timestamp);
}

void blinkLED(int times, int duration) {
  for (int i = 0; i < times; i++) {
    digitalWrite(LED_PIN, HIGH);
    delay(duration);
    digitalWrite(LED_PIN, LOW);
    delay(duration);
  }
}
```

---

5. Sensor Calibration

5.1 Soil Moisture Calibration

```
1. DRY CALIBRATION:
   - Let sensor dry completely in air
   - Record ADC value (typically 3500-4095)
   - Set as SOIL_DRY

2. WET CALIBRATION:
   - Place sensor in glass of water
   - Record ADC value (typically 1200-1800)
   - Set as SOIL_WET

3. FIELD CALIBRATION (recommended):
   - Take known soil samples at different moisture levels
   - Use gravimetric method to get true moisture %
   - Create calibration curve
```

5.2 DHT22 Notes

- Allow 2 seconds between readings

- Accuracy: ±0.5degC, ±2% RH
- Operating range: -40 to 80degC, 0-100% RH

---

6. Synthetic Data Generator

For demos when hardware isn't available:

```python
# synthetic/data_generator.py

import random
import time
import requests
from datetime import datetime, timedelta
import math

class SyntheticFarmSimulator:
    """
    Generates realistic synthetic sensor data for demo purposes.
    Simulates a farm going through dry spell -> irrigation -> recovery cycle.
    """

    def __init__(self, farm_id: str, api_url: str):
        self.farm_id = farm_id
        self.api_url = api_url

        # Initial state
        self.soil_moisture = 55.0  # Start healthy
        self.base_temp = 32.0
        self.base_humidity = 50.0
        self.days_since_rain = 0
        self.last_irrigation = None

        # Simulation parameters
        self.daily_evaporation = 3.5  # % moisture loss per day
        self.temp_variance = 4.0
        self.humidity_variance = 10.0

    def generate_reading(self) -> dict:
        """Generate a single sensor reading."""

        # Time-of-day effects
        hour = datetime.now().hour
        day_factor = math.sin((hour - 6) * math.pi / 12)  # Peak at noon

        # Temperature varies with time of day
        temperature = self.base_temp + (day_factor * 6) + random.gauss(0, 1)
        temperature = max(20, min(45, temperature))

        # Humidity inversely related to temperature
        humidity = self.base_humidity - (day_factor * 15) + random.gauss(0, 3)
        humidity = max(20, min(90, humidity))

        # Soil moisture decreases over time
        evap_rate = self.daily_evaporation * (1 + day_factor * 0.5)
        self.soil_moisture -= evap_rate / 24  # Hourly loss
```

```python
        # Add some noise
        soil_reading = self.soil_moisture + random.gauss(0, 1.5)
        soil_reading = max(5, min(95, soil_reading))

        return {
            "farm_id": self.farm_id,
            "soil_moisture": round(soil_reading, 1),
            "temperature": round(temperature, 1),
            "humidity": round(humidity, 1),
            "timestamp": datetime.utcnow().isoformat() + "Z"
        }

    def simulate_rain(self, mm: float):
        """Simulate rainfall event."""
        moisture_gain = mm * 0.8  # 80% of rain reaches soil
        self.soil_moisture = min(95, self.soil_moisture + moisture_gain)
        self.days_since_rain = 0
        print(f"Rain event: {mm}mm -> Soil moisture now {self.soil_moisture:.1f}%")

    def simulate_irrigation(self, mm: float):
        """Simulate irrigation event."""
        moisture_gain = mm * 0.9  # 90% efficiency
        self.soil_moisture = min(95, self.soil_moisture + moisture_gain)
        self.last_irrigation = datetime.now()
        print(f"Irrigation: {mm}mm -> Soil moisture now {self.soil_moisture:.1f}%")

    def send_to_api(self, reading: dict) -> bool:
        """Send reading to backend API."""
        try:
            response = requests.post(
                self.api_url,
                json=reading,
                headers={"Content-Type": "application/json"},
                timeout=10
            )
            return response.status_code in [200, 201]
        except Exception as e:
            print(f"API Error: {e}")
            return False

    def run_continuous(self, interval_seconds: int = 60):
        """Run continuous data generation."""
        print(f"Starting synthetic data generation for farm: {self.farm_id}")
        print(f"Sending to: {self.api_url}")
        print(f"Interval: {interval_seconds}s")
        print("-" * 50)

        readings_sent = 0

        while True:
            reading = self.generate_reading()

            print(f"[{reading['timestamp']}] "
                  f"Soil: {reading['soil_moisture']}% | "
                  f"Temp: {reading['temperature']}degC | "
                  f"Humidity: {reading['humidity']}%")
```

```python
            if self.send_to_api(reading):
                readings_sent += 1
                print(f"  [x] Sent ({readings_sent} total)")
            else:
                print(f"  [ ] Failed to send")

            # Random events (for demo variety)
            if random.random() < 0.02:  # 2% chance per interval
                self.simulate_rain(random.uniform(5, 25))

            time.sleep(interval_seconds)

    def generate_historical_data(self, days: int = 14) -> list:
        """Generate historical data for seeding the database."""
        data = []
        current_time = datetime.utcnow() - timedelta(days=days)

        # Reset state
        self.soil_moisture = 70.0

        for day in range(days):
            for hour in range(0, 24, 3):  # Every 3 hours
                # Advance time
                timestamp = current_time + timedelta(days=day, hours=hour)

                # Generate reading
                reading = self.generate_reading()
                reading['timestamp'] = timestamp.isoformat() + "Z"
                data.append(reading)

                # Occasional rain
                if random.random() < 0.1:
                    self.simulate_rain(random.uniform(5, 20))

        return data


# Demo scenarios for hackathon
class DemoScenarios:
    """Pre-built scenarios for hackathon demo."""

    @staticmethod
    def healthy_farm(farm_id: str) -> list:
        """Generate data showing healthy conditions."""
        return [
            {"farm_id": farm_id, "soil_moisture": 65, "temperature": 28, "humidity": 60},
            {"farm_id": farm_id, "soil_moisture": 62, "temperature": 30, "humidity": 55},
            {"farm_id": farm_id, "soil_moisture": 60, "temperature": 32, "humidity": 50},
        ]

    @staticmethod
    def drought_progression(farm_id: str) -> list:
        """Generate data showing drought stress developing."""
        data = []
        moisture = 55
        for day in range(14):
            moisture = max(15, moisture - 3 + random.gauss(0, 0.5))
```

```python
            data.append({
                "farm_id": farm_id,
                "soil_moisture": round(moisture, 1),
                "temperature": round(32 + day * 0.3 + random.gauss(0, 1), 1),
                "humidity": round(50 - day * 1.5 + random.gauss(0, 2), 1),
                "timestamp": (datetime.utcnow() - timedelta(days=14-day)).isoformat() + "Z"
            })
        return data

    @staticmethod
    def recovery_after_irrigation(farm_id: str) -> list:
        """Generate data showing recovery after irrigation."""
        data = []
        moisture = 25  # Start low

        # Day 0: Irrigation event
        moisture = 70

        for day in range(7):
            moisture = max(35, moisture - 4 + random.gauss(0, 0.5))
            data.append({
                "farm_id": farm_id,
                "soil_moisture": round(moisture, 1),
                "temperature": round(30 + random.gauss(0, 2), 1),
                "humidity": round(55 + random.gauss(0, 3), 1),
                "timestamp": (datetime.utcnow() - timedelta(days=7-day)).isoformat() + "Z"
            })
        return data


if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser(description="Demeter Synthetic Data Generator")
    parser.add_argument("--farm-id", default="demo-farm-001", help="Farm ID")
    parser.add_argument("--api-url", default="http://localhost:8080/api/v1/sensor-data", help="API endpoint
    parser.add_argument("--interval", type=int, default=60, help="Seconds between readings")
    parser.add_argument("--historical", action="store_true", help="Generate historical data")

    args = parser.parse_args()

    simulator = SyntheticFarmSimulator(args.farm_id, args.api_url)

    if args.historical:
        print("Generating 14 days of historical data...")
        data = simulator.generate_historical_data(14)
        for reading in data:
            simulator.send_to_api(reading)
            print(f"Sent: {reading['timestamp']}")
        print(f"Done! Sent {len(data)} readings.")
    else:
        simulator.run_continuous(args.interval)
```

---

7. Power Management

Battery Operation

```
// Add to ESP32 firmware for solar/battery operation

#include <esp_sleep.h>

#define SLEEP_DURATION_US (5 * 60 * 1000000ULL)  // 5 minutes

void enterDeepSleep() {
  Serial.println("Entering deep sleep...");
  esp_sleep_enable_timer_wakeup(SLEEP_DURATION_US);
  esp_deep_sleep_start();
}

// In loop(), replace delay with:
void loop() {
  sendSensorData();
  enterDeepSleep();  // Wake up after 5 minutes
}
```

Power Budget

State	Current	Duration	Energy
Active + WiFi	150mA	10s	0.42mAh
Deep Sleep	10µA	290s	0.81mAh
Per Hour	-	-	**1.2mAh**

With 2000mAh battery: ~1600 hours = **66 days**

---

8. Enclosure Design

```
+-----------------------------------+
|           JUNCTION BOX            |
|          (IP65 Rated)             |
+-----------------------------------+
|                                   |
|   +---------+      +-----------+  |
|   | ESP32   |      | Battery   |  |
|   +---------+      | Pack      |  |
|                    +-----------+  |
|                                   |
|   Cable gland for sensor wires    |
|         |                         |
+---------+-------------------------+
          |
          | (Waterproof cable)
          |
    +-----+-----+
    |  DHT22    |  (under shade/shield)
    +-----------+
          |
          | (buried in soil)
          |
```

```
    +-----+-----+
    | Soil Sensor|
    +-----------+
```

---

9. Project Structure

```
hardware/
||| demeter_sensor/
|   ||| demeter_sensor.ino      # Main firmware
|   ||| config.h                # WiFi & API config (gitignored)
|   ||| config.example.h        # Template config
||| synthetic/
|   ||| data_generator.py       # Python synthetic data
|   ||| requirements.txt
||| docs/
|   ||| wiring_diagram.png
|   ||| calibration_guide.md
||| pcb/                        # Optional custom PCB
|   ||| demeter_node.kicad
||| README.md
```

---

10. Testing Checklist

Hardware

- ☐ DHT22 reads temperature accurately (±1degC of known reference)
- ☐ DHT22 reads humidity accurately (±5% of known reference)
- ☐ Soil sensor responds to moisture changes
- ☐ Soil sensor calibrated for dry and wet extremes
- ☐ WiFi connects reliably
- ☐ Data posts to API successfully
- ☐ LED indicators working
- ☐ Device recovers from WiFi disconnection
- ☐ Deep sleep wakes correctly (if using battery)

Synthetic Generator

- ☐ Generates realistic data patterns
- ☐ Time-of-day variations working
- ☐ Drought scenario produces declining moisture
- ☐ API integration working
- ☐ Historical data generation working
- ☐ Demo scenarios ready

---

11. Deliverables

- ☐ Working ESP32 sensor node (or well-documented prototype)
- ☐ Synthetic data generator (Python)
- ☐ Demo scenarios for presentation
- ☐ 14 days of historical data seeded
- ☐ Calibration documentation
- ☐ Wiring diagram
- ☐ Parts list / BOM
- ☐ Setup instructions

---

12. Demo Fallback Plan

If hardware fails during demo:

1. **Run synthetic generator** in background
2. **Pre-seed database** with realistic historical data
3. **Use demo scenarios** that show drought progression
4. **Show hardware video** if physical demo not possible

```bash
# Quick demo data generation
python synthetic/data_generator.py \
  --farm-id "demo-farm" \
  --api-url "https://your-api.railway.app/api/v1/sensor-data" \
  --interval 5  # Fast for demo
```

Coordinate with: Backend (API endpoint), AI Engineer (data format expectations)