**Mini-project #3: Data Upload  
Using Bluetooth**

**TEAM MEMBERS**

**RAJ VARSHITH – 16325723**

**SHIVA REDDY – 16352875**

**KARTHIKEYA – 16354793**

**Introduction**

This project uses an **ESP32** microcontroller to read temperature and humidity data from a **DHT11** sensor and broadcast it over **Bluetooth Low Energy (BLE)**. The data is transmitted via BLE characteristics that support notification. Additionally, a simulated battery level characteristic is implemented, which decreases over time to demonstrate power drain. The BLE device is named **"Team 7"**, and the project adheres to BLE standards for transmitting environmental sensor data.

**Components Required:**

* ESP32 Development Board
* DHT11 Temperature and Humidity Sensor
* Jumper wires
* Breadboard (optional)
* BLE Debugging App (e.g., nRF Connect or LightBlue)

**Libraries Used:**

1. **BLEDevice.h**: Manages BLE functionality on the ESP32.
2. **DHT.h**: For interfacing with the DHT11 sensor.
3. **Wire.h**: Enables communication over I2C (if needed for future expansions).

**Project Setup**

**Hardware Connections:**

1. **Connect the DHT22 sensor:**
   * **VCC** to **3.3V** on the ESP32
   * **GND** to **GND** on the ESP32
   * **Data Pin** to **GPIO 14** on the ESP32

**Software Setup:**

1. **Install Arduino IDE**: Ensure the latest version of the Arduino IDE is installed.
2. **Install Required Libraries**:
   * Install the DHT sensor library using the Arduino Library Manager.
   * Install the BLE library for ESP32.
3. **Install BLE Debugging App**:
   * Install **nRF Connect** or **LightBlue** on your mobile phone to verify BLE services and characteristics.

**Code Overview**

**1. Initialization**

BLEDevice::init("Team 7");

* The BLEDevice::init function initializes the BLE device and sets the device name to **"Team 7"**.

**2. BLE Services and Characteristics**

* The project implements the **Environmental Sensing Service** with two characteristics: **Temperature** and **Humidity**. Additionally, a **Battery Service** is created to simulate battery level.

**BLE UUIDs:**

* **Environmental Sensing Service UUID**: 0x181A
* **Temperature Characteristic UUID**: 0x2A6E
* **Humidity Characteristic UUID**: 0x2A6F
* **Battery Service UUID**: 0x180F
* **Battery Level Characteristic UUID**: 0x2A19

BLEService \*envService = pServer->createService(ENVIRONMENTAL\_SENSING\_SERVICE\_UUID);

BLECharacteristic \*temperatureCharacteristic = envService->createCharacteristic(TEMPERATURE\_CHARACTERISTIC\_UUID, BLECharacteristic::PROPERTY\_NOTIFY);

BLECharacteristic \*humidityCharacteristic = envService->createCharacteristic(HUMIDITY\_CHARACTERISTIC\_UUID, BLECharacteristic::PROPERTY\_NOTIFY);

* The **Environmental Sensing Service** contains the temperature and humidity characteristics, both set to support notification properties.

**3. DHT11 Sensor Integration**

float temperature = dht.readTemperature();

float humidity = dht.readHumidity();

* The **DHT11** sensor reads temperature and humidity, and the values are then converted to 16-bit integers for transmission over BLE.

**4. Encoding Data for BLE**

* Temperature and humidity values are scaled by 100 to account for two decimal places.

int16\_t temperatureBLE = (int16\_t)(temperature \* 100);

uint16\_t humidityBLE = (uint16\_t)(humidity \* 100);

* The sensor values are then notified to the client:

temperatureCharacteristic->setValue((uint8\_t\*)&temperatureBLE, sizeof(int16\_t));

humidityCharacteristic->setValue((uint8\_t\*)&humidityBLE, sizeof(uint16\_t));

temperatureCharacteristic->notify();

humidityCharacteristic->notify();

* BLE clients receive real-time updates for temperature and humidity through notifications.

**5. Battery Simulation**

if (batteryLevel > 0) {

batteryLevel--;

batteryCharacteristic->setValue(&batteryLevel, 1);

batteryCharacteristic->notify();

}

* A simple simulation of battery drain reduces the battery level by 1% every minute, notifying connected BLE clients of the updated battery status.

**6. Connection Handling**

class MyServerCallbacks : public BLEServerCallbacks {

void onConnect(BLEServer\* pServer) { deviceConnected = true; }

void onDisconnect(BLEServer\* pServer) { deviceConnected = false; pServer->startAdvertising(); }

};

* The **MyServerCallbacks** class handles BLE connections and disconnections. When the device disconnects, it automatically starts advertising again to allow new connections.

**7. Advertising**

pServer->getAdvertising()->start();

Serial.println("Waiting for a client connection to notify...");

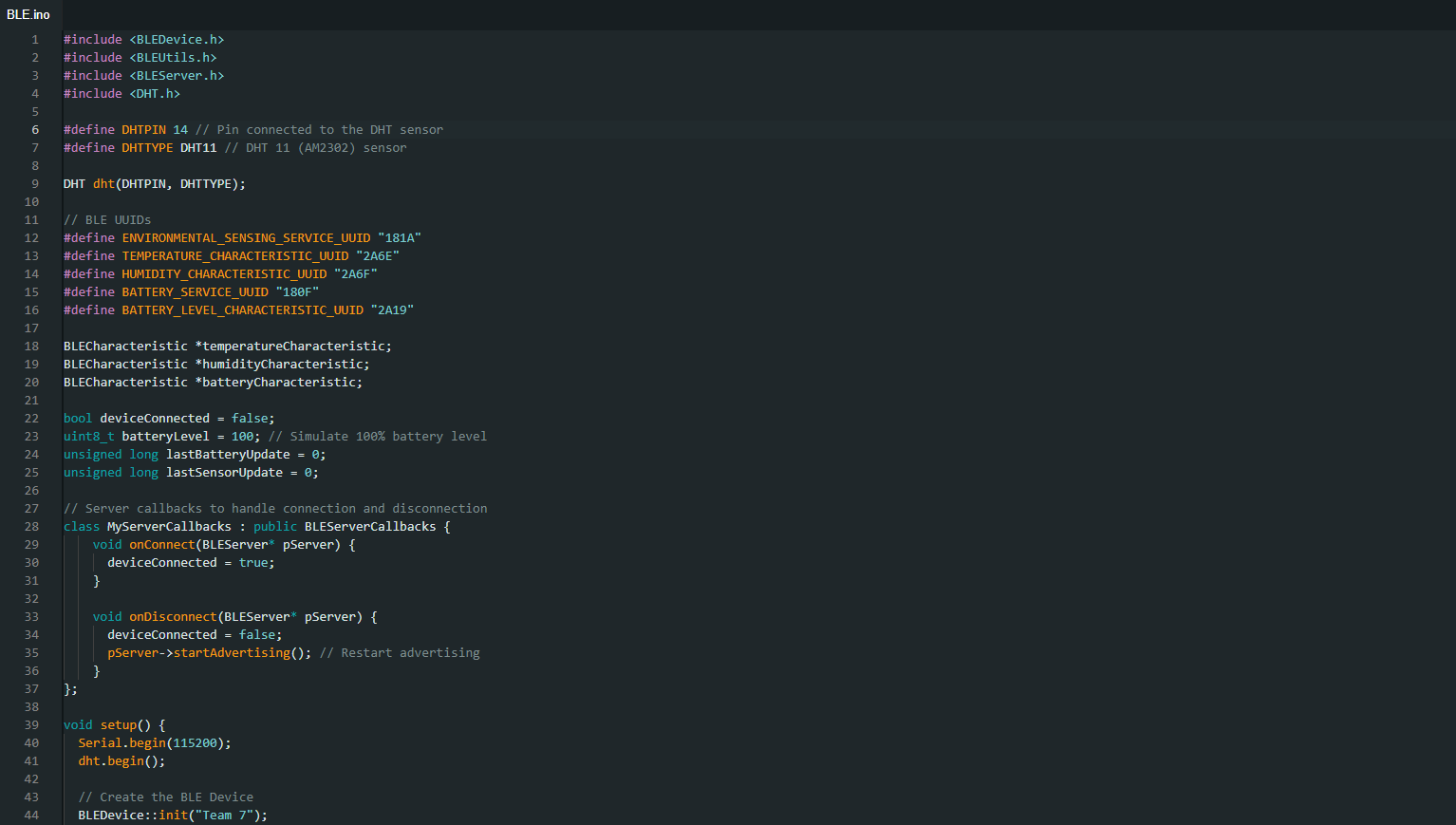
* The ESP32 begins advertising itself as a BLE device after setting up the services and characteristics.

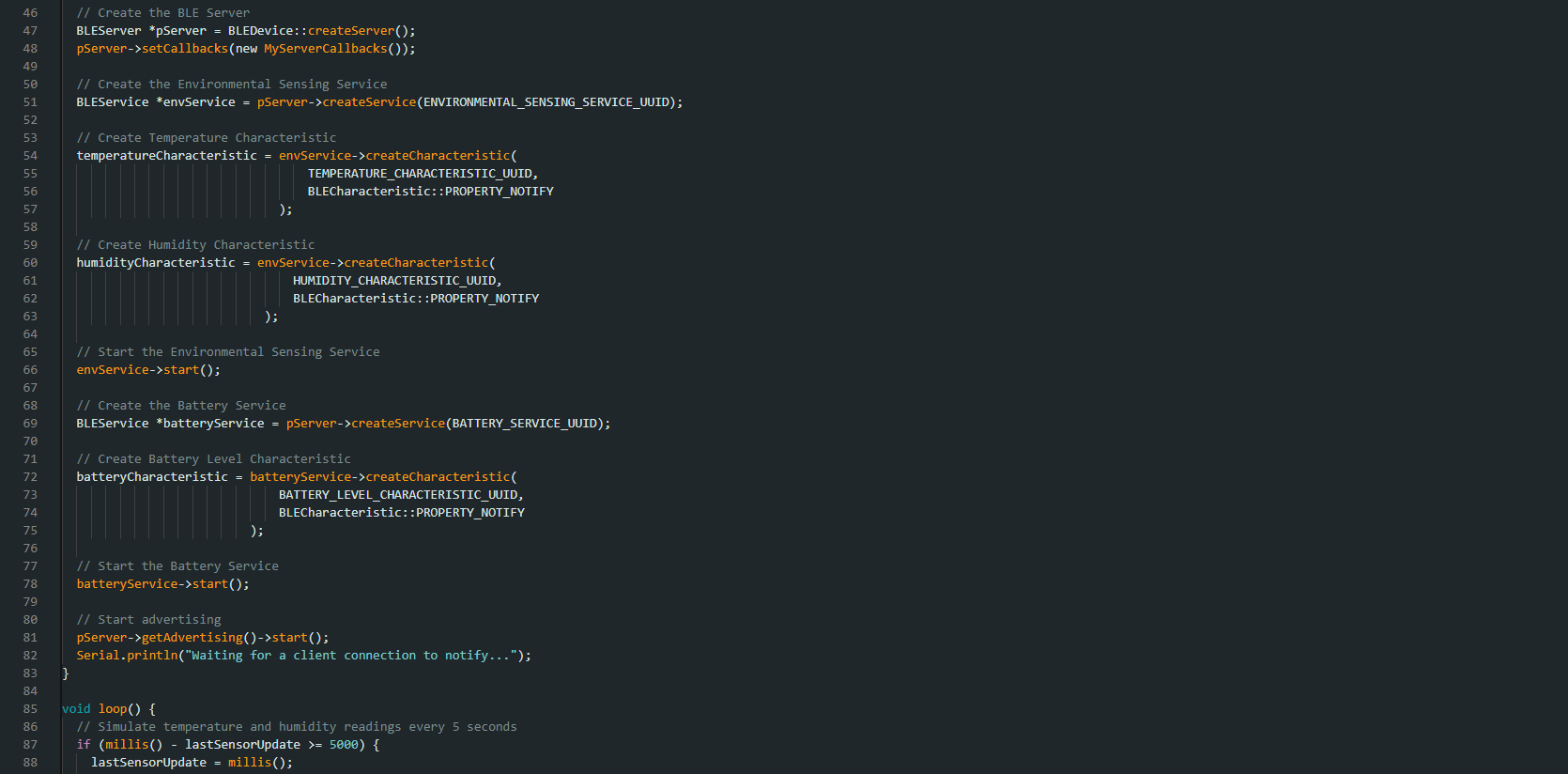
**Testing**

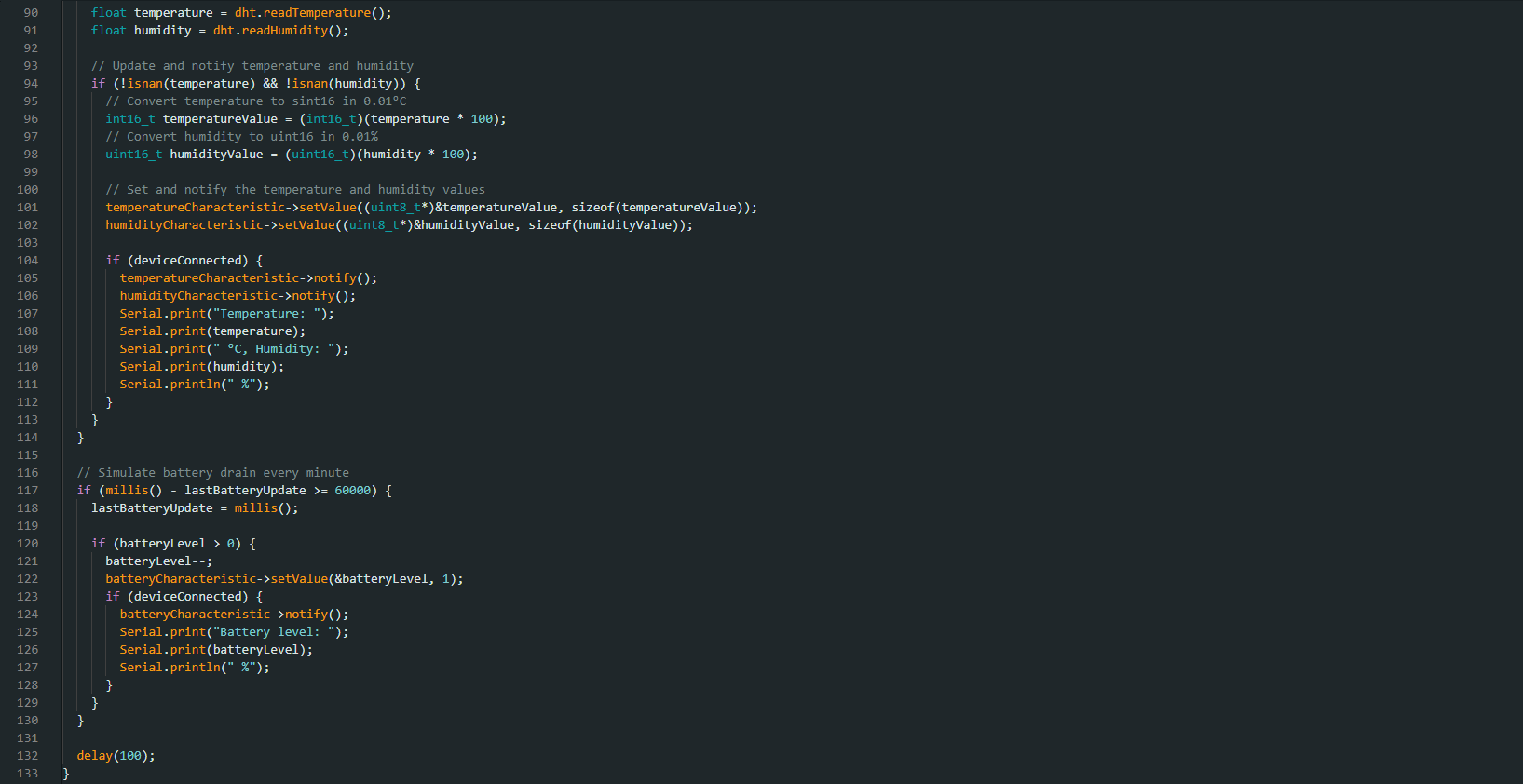
**Steps:**

1. **Upload Code to ESP32**: Flash the ESP32 with the provided code.
2. **Open BLE Debugging App**: Use an app like **nRF Connect** to find **"Team 7"** and connect.
3. **Verify Notifications**: Ensure that the temperature, humidity, and battery level notifications are being received on the mobile app.
4. **Check Serial Monitor**: Use the Arduino IDE Serial Monitor to observe real-time readings of temperature, humidity, and battery level.

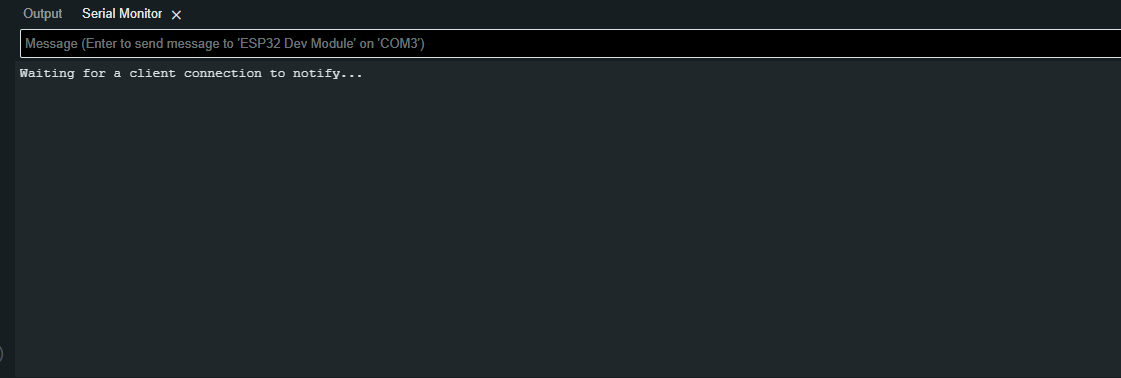
**Code Screenshots:**

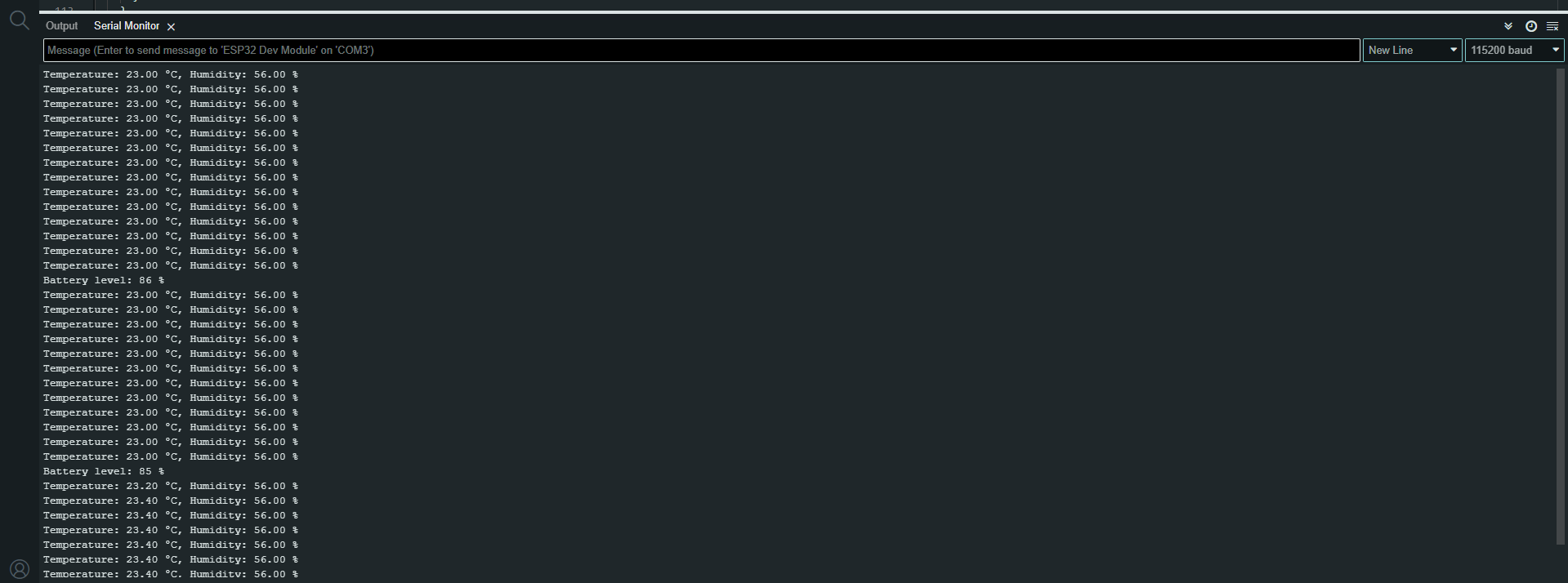






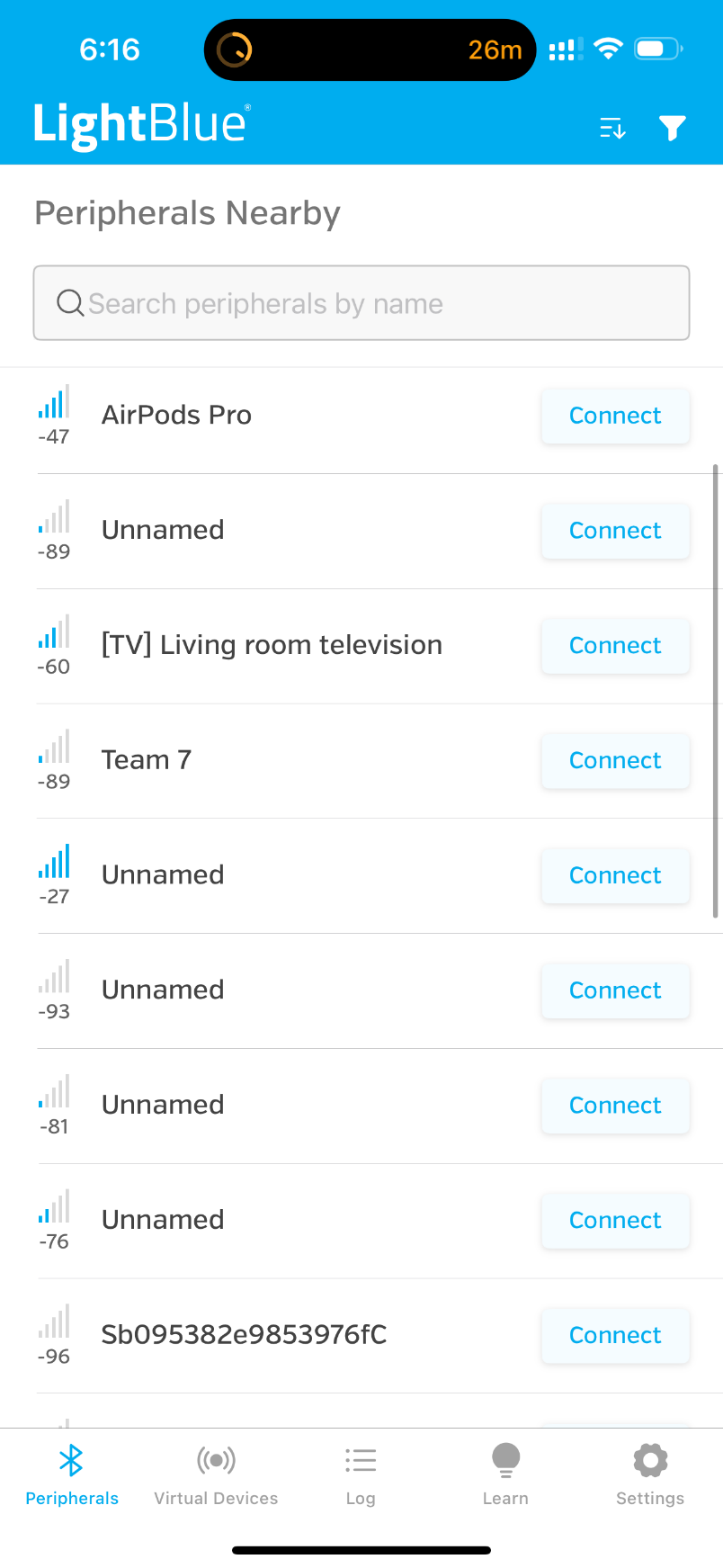
**Outputs:**



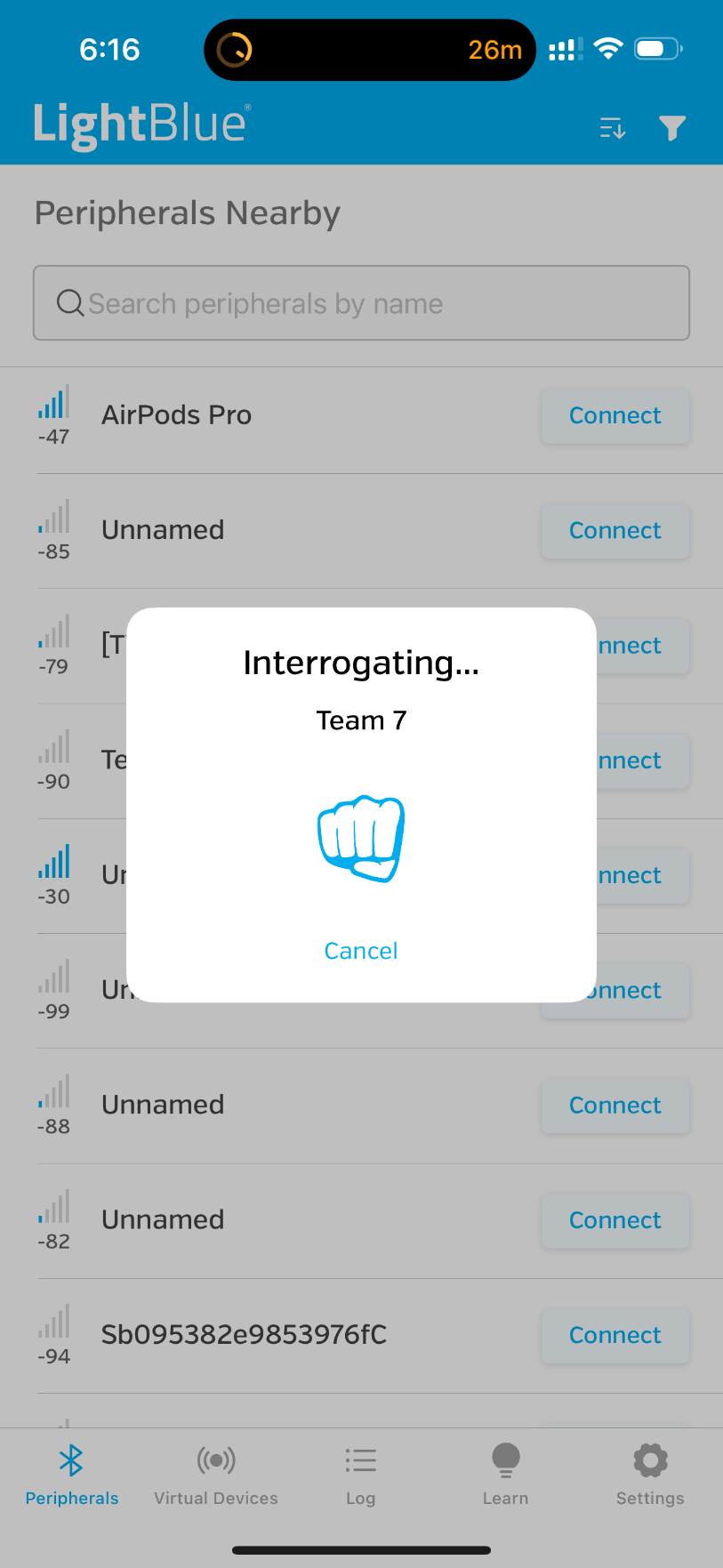


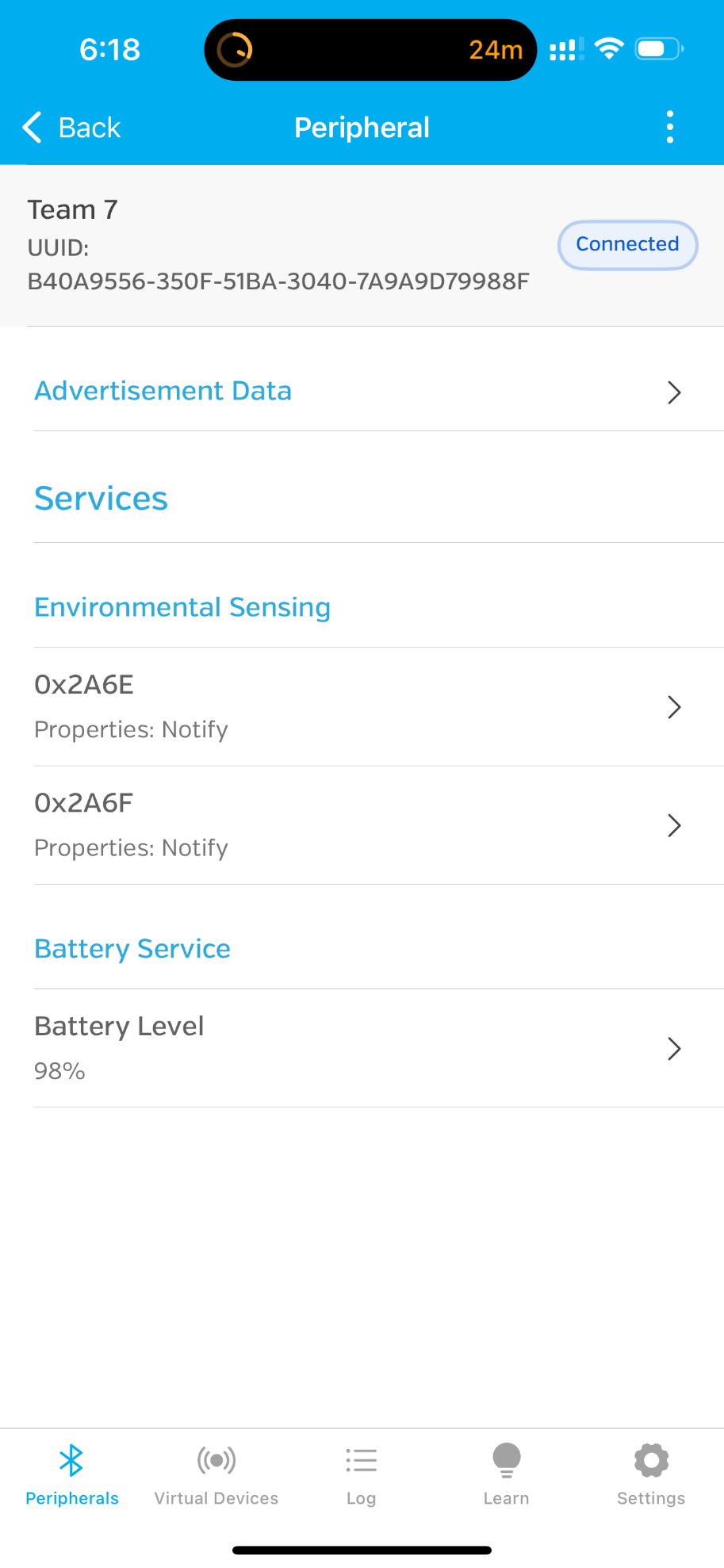
**Output Screenshots from App:**

Team 7:

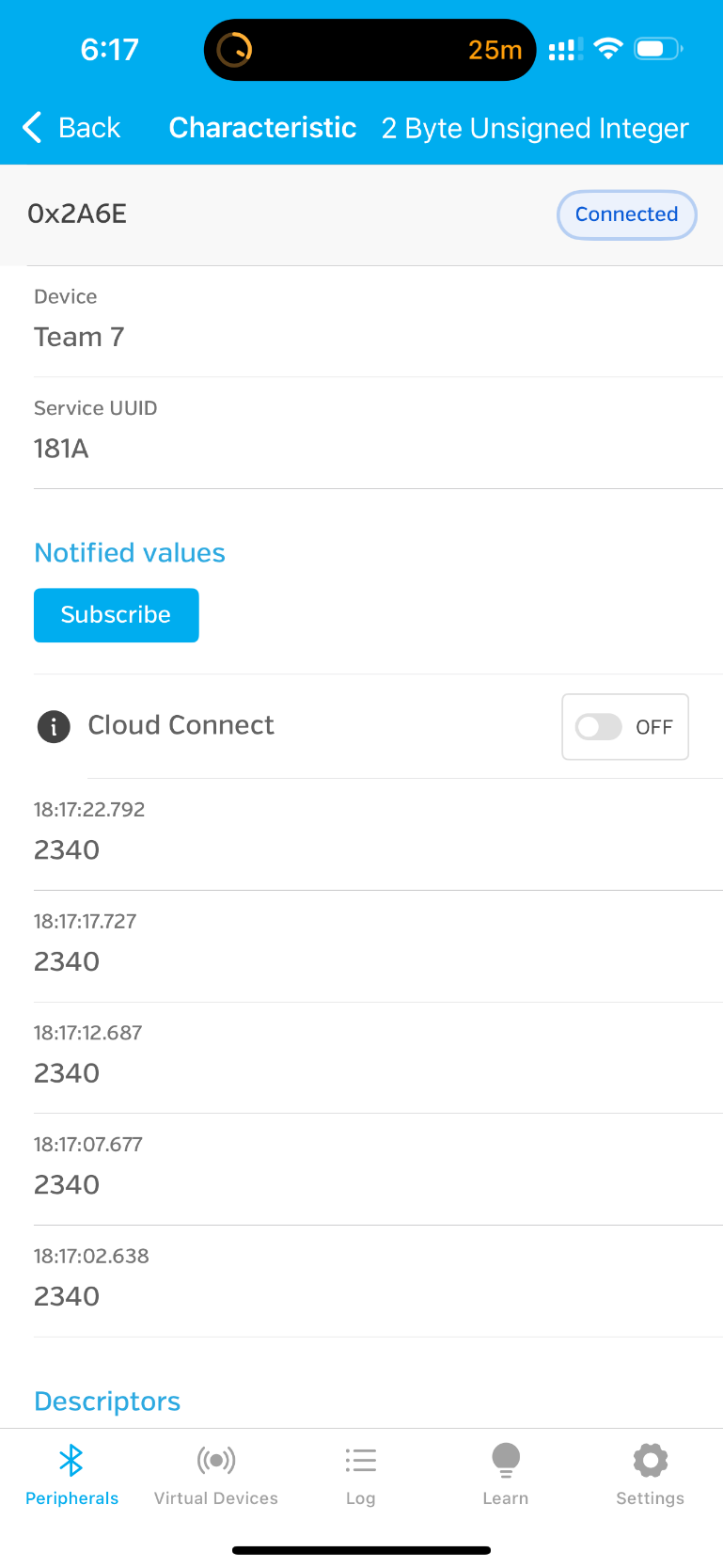
****

**Connecting to the Device**

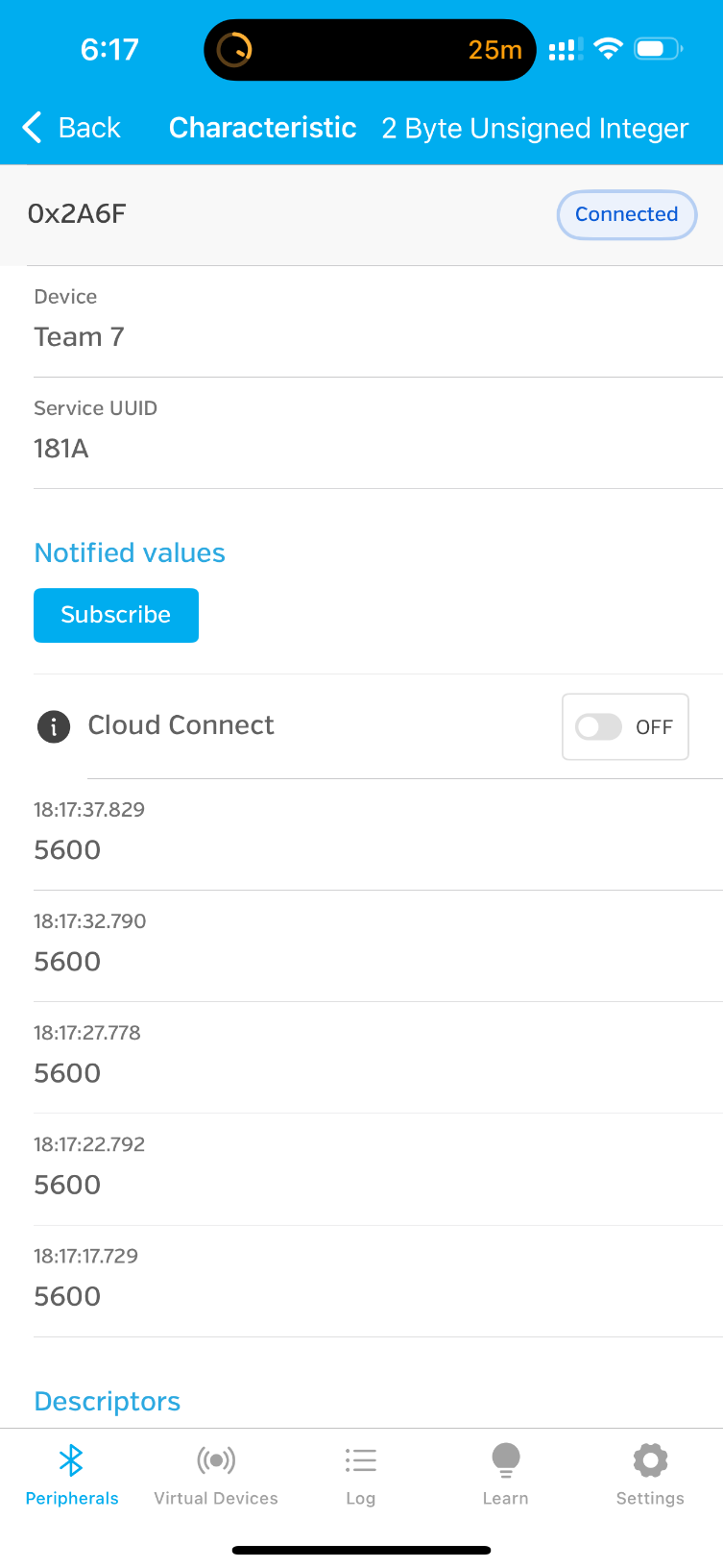




**Reading Temperature:**



**Reading Humidity:**



**Reading the Battery Percentage:**

