Combined Sensors IoT Project

1. Introduction:

This project involves using a Bolt IoT device and various sensors to monitor temperature and light levels. The system will trigger alerts via SMS and control an LED based on predefined threshold values.

2. Components:

- Bolt WiFi Module
- Bolt Cloud Account
- Bolt Smartphone App
- USB Cable
- Temperature Sensor
- Light Sensor (LDR)
- Push Button Switch
- LED x 2
- Buzzer
- Resistor x 2
- Assorted Connecting Wires
- Breadboard

3. Project Goals:

1. **Sensor Integration:**
    - Integrate a Temperature Sensor and a Light Sensor (LDR) with the Bolt IoT device.
    - Establish connections between the sensors and the Bolt WiFi Module.
2. **Continuous Monitoring:**
    - Continuously monitor the real-time values of temperature and light intensity.
3. **Threshold-based Alert System:**
    - Implement a threshold-based alert system for both temperature and light levels.
    - Define predefined threshold values for temperature and light intensity.
4. **Alert Notification:**
    - Trigger an alert system when the observed values exceed or fall below the specified thresholds.
    - Utilize SMS alerts for immediate notification to the user.
5. **Visual Indication:**
    - Provide a visual indication of alerts by controlling an LED.
    - Turn on the LED when an alert is triggered and turn it off after a specified duration.
6. **User Customization:**
    - Allow users to customize threshold values based on their specific requirements.
    - Enable users to adjust the monitoring frequency and LED control duration.
7. **Documentation:**

- Provide detailed step-by-step documentation for the project setup, including hardware connections, software configuration, and code explanation.

8. **User Interaction:**
    - Encourage user interaction by allowing adjustments to key parameters such as sleep duration and threshold values.
    - Implement a user-friendly and informative interface for easy understanding.
9. **Reliability and Robustness:**
    - Ensure the reliability and robustness of the system by handling potential errors and exceptions gracefully.
    - Create a system that operates seamlessly for extended periods without manual intervention.
10. **Educational Value:**
    - Serve as an educational project to enhance understanding of IoT concepts, sensor integration, alert systems, and hardware-software interaction.
11. **Adaptability:**
    - Design the project in a way that allows for future adaptations and enhancements.
    - Encourage users to explore additional features and integrate more sensors if desired.

4. Connections:

4.1. Wiring the Temperature Sensor:

Temperature Sensor Pins:

The temperature sensor usually has three pins: *VCC*, *GND*, and *SIGNAL*.

- Connect the VCC pin to the 5V on the Bolt WiFi Module.
- Connect the GND pin to the GND on the Bolt WiFi Module.
- Connect the SIGNAL pin to any analog pin on the Bolt WiFi Module (let's use A0).

Wiring Steps for Temperature Sensor:

- Connect a jumper wire from the VCC pin of the temperature sensor to the 5V rail on the breadboard.
- Connect another jumper wire from the GND pin of the temperature sensor to the GND rail on the breadboard.
- Connect a third jumper wire from the SIGNAL pin of the temperature sensor to any available slot on the breadboard.
- Connect the other end of the jumper wire from the SIGNAL pin to the A0 pin on the Bolt WiFi Module.

4.2. Wiring the Light Sensor:

Light Sensor Pins:

- The light sensor typically has two pins: VCC, GND, and SIGNAL.
- Connect the VCC pin to the 5V on the Bolt WiFi Module through the breadboard.
- Connect the SIGNAL pin to any other analog pin on the Bolt WiFi Module (let's use A0).
- Connect one leg of the resistor to analog pin A0 and the other leg in GND.

Wiring Steps for Light Sensor:

- Connect a jumper wire from the VCC pin of the light sensor to the 5V rail on the breadboard.
- Connect a second jumper wire from the SIGNAL pin of the light sensor to analog pin A0.
- Connect another jumper wire from the GND pin of the resistor to the GND rail on the breadboard and also the another leg in analog pin A0.

4.3. Wiring the LED for Alert:

LED Pins:

- The LED has two pins: Anode (longer pin) and Cathode (shorter pin).
- Connect the Anode to a current-limiting resistor (around 220-330 ohms).
- Connect the other end of the resistor to any digital pin on the Bolt WiFi Module (let's use 0).
- Connect the Cathode directly to the GND on the Bolt WiFi Module.

Wiring Steps for LED:

- Connect the Anode of the LED to one leg of the resistor.
- Connect the other leg of the resistor to any available slot on the breadboard.
- Connect a jumper wire from the same slot to the 0 pin on the Bolt WiFi Module.
- Connect the Cathode of the LED directly to the GND rail on the breadboard.

Summary of Connections:

Temperature Sensor:

VCC --> 5V rail on breadboard

GND --> GND rail on breadboard

SIGNAL --> A0 on Bolt WiFi Module

Light Sensor:

VCC --> 5V rail on the breadboard

A0 --> A0 rail on breadboard

RESISTOR (10k ohms)-> A0 on breadboard and GND rail on breadboard

LED for Alert:

Anode --> Resistor --> 0 on Bolt WiFi Module

Cathode --> GND

- Make all necessary power and ground connections.

5. Code Overview:

- Firstly, create a file called ‘conf.py’.

5.1. conf.py:

```
SID = 'AC93463c364e04e09e47b92aaa2291c5bd'
AUTH_TOKEN = '5f5bf33ec7fe151ff784178b55ec3a08'
FROM_NUMBER ='+12064721661'
TO_NUMBER ='+918639470630'
API_KEY = 'aa30ba6d-e7c1-4dfe-b594-cc3186ddddf0'
DEVICE_ID = 'BOLT6337362'
```

- After writing the details save the file.

5.2. combined_sensors.py:

Now, Write the actual code in the file name called ‘combined_sensors.py’

5.2.1. Importing Libraries:

```
import conf
from boltiot import Bolt, Sms
import json # Import the json module
import time
```

1. **import conf**: This line imports the **conf** module. In your case, it's likely a separate Python file (**conf.py**) where you store sensitive information such as API keys and credentials. This allows you to keep your sensitive data separate from the main script for security reasons.
2. **from boltiot import Bolt, Sms**: This line imports the **Bolt** and **Sms** classes from the **boltiot** module. These classes provide functionality for interacting with the Bolt Cloud and sending SMS alerts using Twilio.
3. **import json**: This line imports the **json** module, which provides methods for working with JSON data. In your script, it's used to convert JSON strings to Python dictionaries.
4. **import time**: This line imports the **time** module, which provides various time-related functions. In your script, it's used for introducing delays, such as waiting between sensor readings.

5.2.2. Bolt and SMS Object Initialization:

```
# Create Bolt and Sms objects
mybolt = Bolt(conf.API_KEY, conf.DEVICE_ID)
sms = Sms(conf.SID, conf.AUTH_TOKEN, conf.TO_NUMBER, conf.FROM_NUMBER)
```

1. These lines of code create instances of the **Bolt** and **Sms** classes, initializing them with the required parameters such as API key, device ID, Twilio credentials, and phone numbers. This step is crucial for establishing connections to the Bolt Cloud for hardware communication and for sending SMS alerts.
2. The **Sms** object, named **sms**, is created using the **Sms** class. This object is initialized with Twilio credentials (SID and Auth Token), as well as the recipient and sender phone numbers. The **sms** object will be utilized for sending alert messages via SMS when specific conditions are met.

5.2.3. Threshold Configuration:

```
TEMP_THRESHOLD_HIGH = 20  # Set your desired temperature threshold
TEMP_THRESHOLD_LOW = 10
LIGHT_THRESHOLD_HIGH = 800  # Set your desired light threshold
LIGHT_THRESHOLD_LOW = 200
```

1. **TEMP_THRESHOLD_HIGH:**
    - This threshold represents the upper limit for the temperature sensor. When the temperature reading goes beyond this limit, the system will generate a temperature alert.

2. **TEMP_THRESHOLD_LOW:**
    - This threshold represents the lower limit for the temperature sensor. When the temperature reading goes below this limit, the system will generate a temperature alert.
3. **LIGHT_THRESHOLD_HIGH:**
    - This threshold represents the upper limit for the light sensor. When the light intensity reading exceeds this limit, the system will generate a light intensity alert.
4. **LIGHT_THRESHOLD_LOW:**
    - This threshold represents the lower limit for the light sensor. When the light intensity reading goes below this limit, the system will generate a light intensity alert.

5.2.4. Pin Configuration and Sensor Reading:

```
# Pin configuration
LED_PIN = 0  # Assuming pin 0 functions as a digital pin

# Function to check sensor values and trigger alert
def check_and_alert():
    # Read sensor values
    temperature_response = mybolt.analogRead('A0')
    light_response = mybolt.analogRead('A0')  # Adjusted to read from A0 for the
light sensor
```

1. It establishes the configuration for the LED pin, assuming that pin 0 on the Bolt WiFi Module functions as a digital pin.
2. The LED pin is crucial for visual alerts, and its value will be used in the code for controlling the LED.

3. Here it defines a function named **check_and_alert** responsible for reading sensor values and triggering alerts.
4. Two sensor readings are obtained: one for the temperature sensor (**A0**) and another for the light sensor (**A0**). The **analogRead** function is used to read analog values from the specified pins.
5. The light sensor reading is adjusted to read from pin **A0** to reflect the correct sensor configuration.
6. These sensor readings are stored in variables (**temperature_response** and **light_response**) for further processing in the code.

5.2.5. Data Conversion from JSON String to Python Dictionaries:

```
# Convert JSON strings to Python dictionaries
  temperature_data = json.loads(temperature_response)
  light_data = json.loads(light_response)
```

JSON (JavaScript Object Notation) is a lightweight data interchange format. The **json.loads** function in Python is used to deserialize a JSON string into a Python dictionary. This conversion is essential because the **analogRead** function of the Bolt library returns the sensor readings in JSON format.

1. **temperature_data**: This variable holds the Python dictionary containing information about the temperature sensor reading. The 'value' key within this dictionary is extracted to obtain the actual temperature value.
2. **light_data**: Similarly, this variable contains the Python dictionary with information about the light sensor reading. The 'value' key is extracted to obtain the light intensity value.

5.2.6. Extracting Analog Readings from JSON Response:

```
# Extract analog readings from the JSON response
    temperature = int(temperature_data['value'])
    light = int(light_data['value'])
```

1. **temperature_data**: This variable contains the data obtained from the analogRead function for the temperature sensor.
2. **['value']**: Accessing the 'value' key in the JSON response, which represents the actual analog reading.
3. **int(temperature_data[‘value’])**: Converting the analog reading from a string to an integer for numerical comparison.
4. **light_data**: Similar to temperature_data, this variable holds the analog reading data for the light sensor.
5. **['value']**: Accessing the 'value' key in the JSON response for the light sensor.
6. **int(light_data[‘value’])**: Converting the analog reading from a string to an integer for numerical comparison

5.2.7. Sensor Threshold Checking:

```
# Check temperature and light levels
        if temperature > TEMP_THRESHOLD_HIGH or temperature < TEMP_THRESHOLD_LOW:
            alert_message = "Temperature Alert! Current Temperature: {} C".format(temperature)
        send_alert(alert_message)

            if light > LIGHT_THRESHOLD_HIGH or light < LIGHT_THRESHOLD_LOW:
        alert_message = "Light Alert! Current Light Intensity: {}".format(light)
        send_alert(alert_message)
```

1. **Temperature Monitoring:**
    - The first **if** statement checks whether the current temperature reading (**temperature**) is either higher than the predefined upper threshold (**TEMP_THRESHOLD_HIGH**) or lower than the predefined lower threshold (**TEMP_THRESHOLD_LOW**).
    - If the condition is true, it triggers a temperature alert.
    - The **alert_message** is then formatted to include the current temperature reading.
2. **Light Intensity Monitoring:**
    - The second **if** statement checks whether the current light intensity reading (**light**) is either higher than the predefined upper threshold (**LIGHT_THRESHOLD_HIGH**) or lower than the predefined lower threshold (**LIGHT_THRESHOLD_LOW**).
    - If the condition is true, it triggers a light intensity alert.
    - Similar to the temperature alert, the **alert_message** is formatted to include the current light intensity reading.

5.2.8. Alert Notification and Visual Indicator:

```
# Function to send alert SMS and control LED
def send_alert(message):
    response = sms.send_sms("Alert: " + message)
    print("SMS Response: ", response)

    # Turn on the LED
    mybolt.digitalWrite(LED_PIN, 'HIGH')
    time.sleep(2)  # Wait for 2 seconds
    # Turn off the LED
    mybolt.digitalWrite(LED_PIN, 'LOW')
    time.sleep(2)
```

This function, **send_alert**, serves the purpose of notifying users about an alert by sending an SMS and providing a visual indicator using an LED. Here's a breakdown of its components:

- **SMS Alert:**
    - The **sms.send_sms** method is used to send an SMS alert with the specified message.
    - The response from the SMS service is printed to the console for monitoring purposes.
- **LED Control:**
    - The LED connected to the **LED_PIN** is turned on by setting the pin to 'HIGH'.
    - A 2-second delay (**time.sleep(2)**) is introduced to keep the LED on for a brief period.
    - Subsequently, the LED is turned off by setting the pin to 'LOW'.
    - Another 2-second delay is added for visual indication.

These actions create a dual feedback system where users receive an SMS alert, and an LED visually indicates the occurrence of an event. The delays ensure that the LED remains on and off for a short duration, enhancing the visibility of the alert signal.

5.2.9. Main Loop for Continuous Monitoring:

```
# Main loop to continuously check and alert
while True:
    check_and_alert()
    time.sleep(5)
```

1. **while True:**: This initiates an infinite loop, ensuring that the code inside the loop will run indefinitely until manually interrupted.
2. **check_and_alert():**: Calls the **check_and_alert** function, which is responsible for reading sensor values, evaluating them against predefined thresholds, and triggering alerts accordingly.
3. **time.sleep(5)**: Introduces a 5-second delay between successive iterations of the loop. This delay is adjustable based on your specific requirements

6. Conclusion:

This project demonstrates the integration of temperature and light sensors with the Bolt IoT device. Alerts are sent via SMS, and visual indicators (LED) are used to notify users when predefined thresholds are exceeded.

7. Running the Code:

1. Connect the Bolt WiFi Module to the sensors as per the documentation.

2. Update the `conf.py` file with your actual credentials.

3. Run the `combined_sensors.py` script using Python.

Now, the system will continuously monitor sensor values and trigger alerts based on the specified thresholds.