프로젝트 제목

김민준, 20235128, 컨텐츠it

목차

I. 서론

1. 프로젝트의 필요성

기존 아파트에 설치되어 있는 사물 인터넷 서비스 기기들은 온도를 제어하거나 불을 끄는 정도의 제어만 제공하는 경우가 다수였습니다. 저는 이를 보안해 방 안에 공기와 외부 공기를 사용자가 원하는 데로 컨트롤 할수 있게 해주는 "스마트 환기 장치"를 만들게 되었습니다.

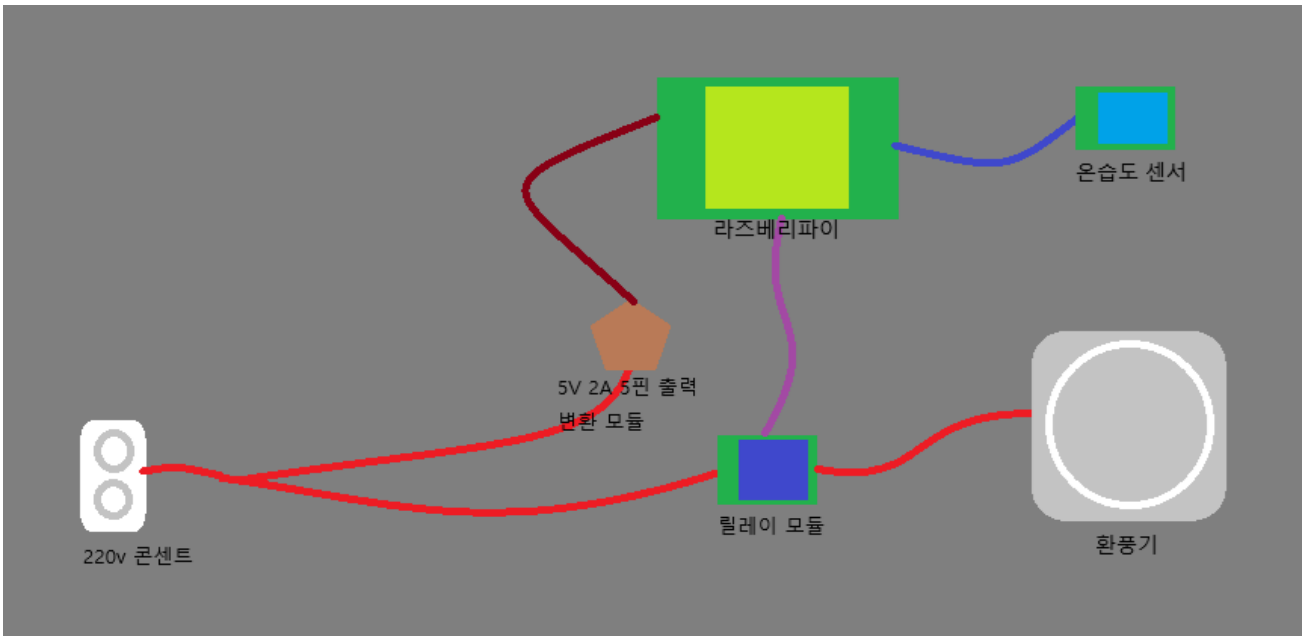
2. 프로젝트의 목적

CoAP 통신을 통해 사용자는 "스마트 환기 장치" 가 설치된 환경 정보를 얻을수 있고, 그에 따라 사용자가 원하는 환기를 즉각적으로 진행할수 있게 원격으로 제어할수 있는 시스템을 구축하는게 그 목적 입니다.

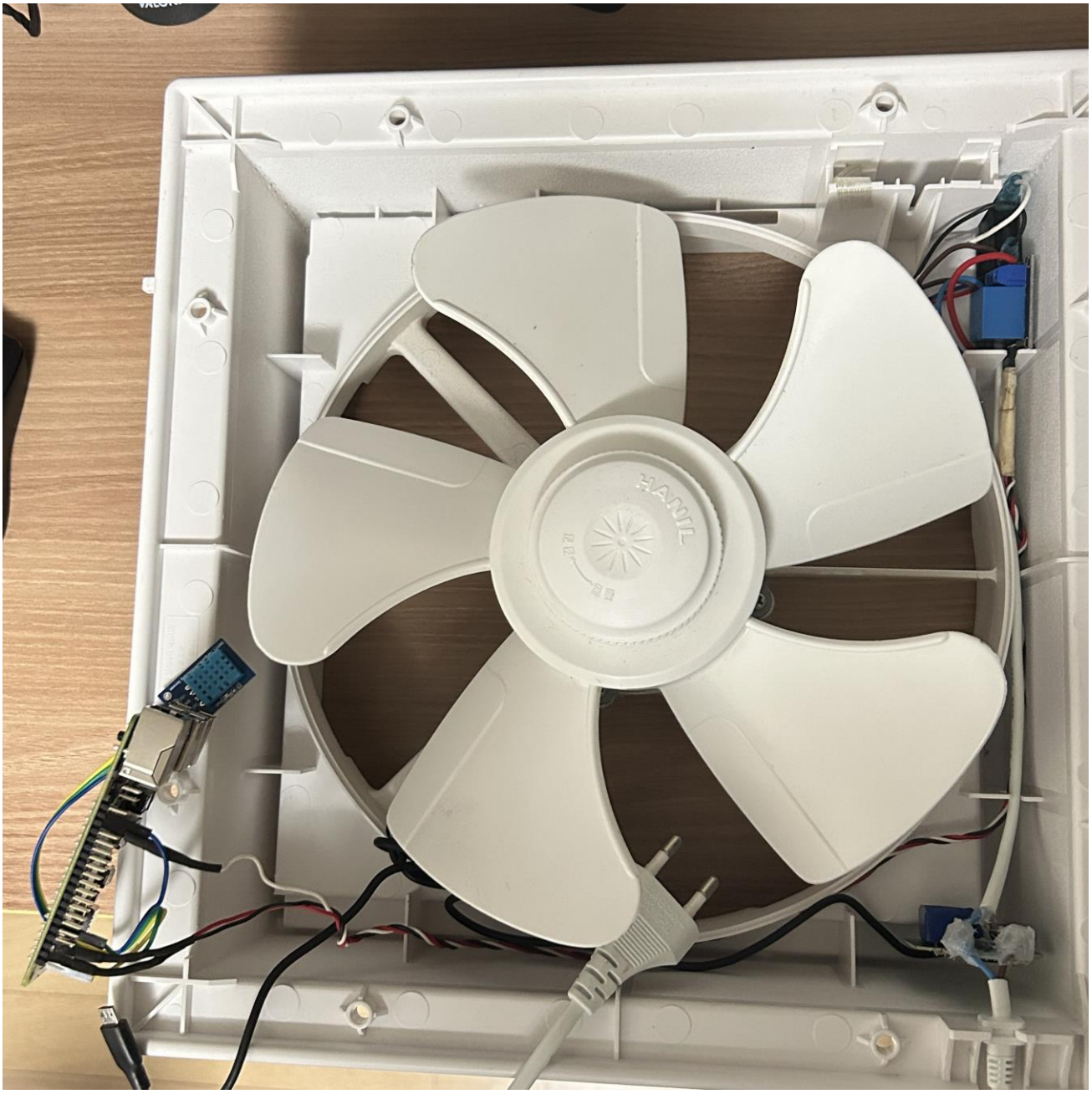
II. 본론

1. 하드웨어 구현

이 장치는 하나의 220v 전원으로 작동 합니다. 회로의 구성은 다음과 같습니다.



전원이 입력되고 라즈베리파이가 네트워크에 연결된후 CoAP 통신으로 작동을 시작합니다. 아래의 사진은 실제 구현 사진 입니다.



온도센서, 릴레이 모듈은 라즈베리파이와 연결되어 있습니다. 라즈베리파이 전원용 전원 변환 모듈과 환풍기 팬의 전원은 220v 입력을 받아 작동합니다.

2. 소프트웨어 구현

1. Sever 소스 코드 설명

1.1 ServerCoAP 클래스

이 클래스는 CoAP 서버를 실행하는 주요 클래스로, 서버를 시작하고 리소스를 등록하는 역할을 합니다.

```java
package ServerCoAP;

import org.ws4d.coap.core.rest.CoapResourceServer;

public class ServerCoAP {
    /* Field */
    private static ServerCoAP coapServer;
    private CoapResourceServer resourceServer;

    /* Method */
    public void start() {
        System.out.println(x:"===Run CoAP Server ===");
        // create server
        if (this.resourceServer != null) {
            this.resourceServer.stop();
        }
        this.resourceServer = new CoapResourceServer();
        // initialize resource
        Relay relay = new Relay();
        DHT11 dht11 = new DHT11();
        // add resource to server
        this.resourceServer.createResource(relay);
        this.resourceServer.createResource(dht11);
        // run the server
        try {
            this.resourceServer.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /* Main */
    Run | Debug
    public static void main(String[] args) {
        coapServer = new ServerCoAP();
        coapServer.start();
    }
}
```

설명:

- `ServerCoAP` 클래스는 서버의 시작을 담당하는 `start()` 메서드를 포함하고 있습니다.
- `CoapResourceServer`는 CoAP 서버를 설정하는 객체입니다.

- 서버가 실행되기 전에 Relay와 DHT11 리소스를 추가하고, 리소스를 서버에 등록한 뒤, 서버를 시작합니다.
- main() 메서드에서 ServerCoAP 객체를 생성하고 start() 메서드를 호출하여 서버를 실행합니다.

1.2 DHT11 클래스

이 클래스는 DHT11 온습도 센서를 읽고 CoAP 클라이언트가 요청했을 때 온도와 습도 정보를 반환하는 역할을 합니다.

```java
package ServerCoAP;

import java.util.List;

import org.ws4d.coap.core.enumerations.CoapMediaType;
import org.ws4d.coap.core.rest.BasicCoapResource;
import org.ws4d.coap.core.rest.CoapData;

public class DHT11 extends BasicCoapResource {
    /* Field */
    private float temperature; // 온도 값
    private float humidity; // 습도 값

    /* Constructor */
    DHT11(String path, String value, CoapMediaType mediaType) {
        super(path, value, mediaType);
    }

    DHT11() {
        this(path:"/dht11", value:"0", CoapMediaType.text_plain);
        this.temperature = 0.0f;
        this.humidity = 0.0f;
    }

    /* Method */
    private void readDHT11() {
        final boolean FOR_TEST = false;
        if(FOR_TEST){
            this.temperature = 25.5f; // 가상의 온도 값
            this.humidity = 60.0f; // 가상의 습도 값
        }else{
            final EasyDHT11 dht = new EasyDHT11();
            float[] read = dht.getData(pin:28); // Read data from GPIO 28
            this.temperature = read[0];
            this.humidity = read[1];
        }
    }
```

```
38
39      @Override
40      public synchronized CoapData get(List<String> query, List<CoapMediaType> mediaTypesAccepted) {
41          return this.get(mediaTypesAccepted);
42      }
43
44      @Override
45      public synchronized CoapData get(List<CoapMediaType> mediaTypesAccepted) {
46          // DHT11 센서 데이터를 읽음
47          this.readDHT11();
48
49          // 온도와 습도 정보를 문자열로 결합하여 반환
50          String data = "Temperature: " + this.temperature + ", Humidity: " + this.humidity;
51          return new CoapData(data.getBytes(), CoapMediaType.text_plain);
52      }
53
54      @Override
55      public synchronized String getResourceType() {
56          return "RaspberryPi-4 DHT11";
57      }
58  }
59
```

설명:

- `DHT11` 클래스는 CoAP 리소스를 구현한 클래스로, `CoapResourceServer`에 등록됩니다.
- `readDHT11()` 메서드는 DHT11 센서에서 데이터를 읽어 `temperature`와 `humidity` 값을 업데이트합니다.
- `get()` 메서드는 CoAP GET 요청을 처리하고 센서 값을 문자열로 변환하여 반환합니다. 요청에 대해 센서 데이터를 포함한 응답을 `CoapData` 객체로 생성해 반환합니다.
- `getResourceType()` 메서드는 리소스의 유형을 반환합니다.

1.3 Relay 클래스

이 클래스는 릴레이 모듈을 제어하고, CoAP 요청을 받아 상태를 변경할 수 있게 합니다.

```java
package ServerCoAP;

import java.util.List;

import org.ws4d.coap.core.enumerations.CoapMediaType;
import org.ws4d.coap.core.rest.BasicCoapResource;
import org.ws4d.coap.core.rest.CoapData;

import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;

public class Relay extends BasicCoapResource {
    /* Field */
    private GpioController gpio;
    private GpioPinDigitalOutput pin;
    private boolean state; // Boolean으로 상태 관리

    /* Constructor */
    Relay(String path, String value, CoapMediaType mediaType) {
        super(path, value, mediaType);
    }

    Relay() {
        this(path:"/relay", value:"off", CoapMediaType.text_plain);
        this.gpio = GpioFactory.getInstance();
        this.pin = this.gpio.provisionDigitalOutputPin(RaspiPin.GPIO_27, PinState.LOW); // Relay 모듈의 GPIO 핀은 27번
        this.state = false; // 초기 상태는 off(false)
    }

    /* Method */
    @Override
    public synchronized CoapData get(List<String> query, List<CoapMediaType> mediaTypesAccepted) {
        return this.get(mediaTypesAccepted);
    }
```

```java
    @Override
    public synchronized CoapData get(List<CoapMediaType> mediaTypesAccepted) {
        // 텍스트 형식으로 반환 (보통 CoAP는 텍스트일 때 string으로 응답하지만, byte로 변환한 후 적절한 미디어 타입 지정)
        String data = state ? "ON" : "OFF";  // boolean 상태를 "ON" 또는 "OFF"로 문자열 변환

        return new CoapData(data.getBytes(), CoapMediaType.text_plain);
    }


    @Override
    public synchronized boolean setValue(byte[] value) {
        // byte[] 값을 boolean으로 변환
        if (value != null && value.length > 0) {
            this.state = value[0] != 0; // 0이 아니면 true
        }

        // Relay 핀 상태 변경
        if (this.state) {
            pin.high(); // Relay ON
        } else {
            pin.low(); // Relay OFF
        }

        return true;
    }

    @Override
    public synchronized boolean post(byte[] data, CoapMediaType type) {
        return this.setValue(data);
    }

    @Override
    public synchronized boolean put(byte[] data, CoapMediaType type) {
        return this.setValue(data);
    }
```

```
74
75      @Override
76      public synchronized String getResourceType() {
77          return "RaspberryPi-4 Relay";
78      }
79  }
80
```

설명:

- `Relay` 클래스는 CoAP 리소스이자 Raspberry Pi의 GPIO 핀을 통해 릴레이 모듈을 제어합니다.
- `get()` 메서드는 릴레이의 상태를 반환합니다 (ON/OFF).
- `setValue()` 메서드는 CoAP PUT 또는 POST 요청을 받아 상태 값을 변경하고 GPIO 핀을 통해 릴레이를 제어합니다.
- `getResourceType()` 메서드는 리소스의 유형을 반환합니다.

1.4 `EasyDHT11` 클래스

이 클래스는 DHT11 센서에서 실제 데이터를 읽어오는 클래스입니다. `getData()` 메서드는 GPIO 핀에서 데이터를 읽고, 체크섬을 확인한 후 온도와 습도를 반환합니다.

```java
package ServerCoAP;

import com.pi4j.wiringpi.Gpio;

public class EasyDHT11 {
    private static final int MAXTIMINGS = 85;
    private final int[] dht11_f = { 0, 0, 0, 0, 0 }; // DHT11 data format (5 bytes)

    public EasyDHT11() {
        // Setup wiringPi
        if (Gpio.wiringPiSetup() == -1) {
            System.out.println(x:" ==>> GPIO SETUP FAILED");
            return;
        }
    }

    public float[] getData(final int pin) {
        int laststate = Gpio.HIGH; // signal 상태 변화를 알기 위해 기존 상태를 기억
        int j = 0; // 수신한 Bit의 index counter
        float h = -99; // 습도
        float c = -99; // 섭씨 온도
        float f = -99; // 화씨 온도

        // Integral RH, Decimal RH, Integral T, Decimal T
        dht11_f[0] = dht11_f[1] = dht11_f[2] = dht11_f[3] = dht11_f[4] = 0;

        // 1. DHT11 센서에게 start signal 전달
        Gpio.pinMode(pin, Gpio.OUTPUT);
        Gpio.digitalWrite(pin, Gpio.LOW);
        Gpio.delay(18);

        // 2. Pull-up -> 수신 모드로 전환 -> 센서의 응답 대기
        Gpio.digitalWrite(pin, Gpio.HIGH);
        Gpio.pinMode(pin, Gpio.INPUT);
```

```java

        // 2. Pull-up -> 수신 모드로 전환 -> 센서의 응답 대기
        Gpio.digitalWrite(pin, Gpio.HIGH);
        Gpio.pinMode(pin, Gpio.INPUT);

        // 3. 센서의 응답에 따른 동작
        for (int i = 0; i < MAXTIMINGS; i++) {
            int counter = 0;
            while (Gpio.digitalRead(pin) == laststate) { // Gpio pin 상태가 바뀌지 않으면 대기
                counter++;
                Gpio.delayMicroseconds(1);
                if (counter == 255) {
                    break;
                }
            }
            laststate = Gpio.digitalRead(pin);
            if (counter == 255) {
                break;
            }

            // 각각의 bit 데이터 저장
            if (i >= 4 && i % 2 == 0) { // 첫 3개의 상태 변화는 무시, laststate가 low에서 high로 바뀔 때만 값을 저장
                // data 저장
                dht11_f[j / 8] <<= 1; // 0 bit
                if (counter > 16) {
                    dht11_f[j / 8] |= 1; // 1 bit
                }
                j++;
            }
        }
```

```
61
62          // Checksum 확인
63          // Check we read 40 bits (8 bit x 5) + verify checksum in the last
64          if (j >= 40 && getChecksum()) {
65              h = (float) ((dht11_f[0] << 8) + dht11_f[1]) / 10;
66              if (h > 100) {
67                  h = dht11_f[0]; // for DHT11
68              }
69              c = (float) (((dht11_f[2] & 0x7F) << 8) + dht11_f[3]) / 10;
70              if (c > 125) {
71                  c = dht11_f[2]; // for DHT11
72              }
73              if ((dht11_f[2] & 0x80) != 0) {
74                  c = -c;
75              }
76              f = c * 1.8f + 32;
77              System.out.println("Humidity = " + h + "% Temperature = " + c + "℃ | " + f + "°F)");
78          }
79          else {
80              System.out.println(x:"Checksum Error");
81          }
82
83          float[] result = { h, c, f };
84          return result;
85      }
86
87      private boolean getChecksum() {
88          return dht11_f[4] == (dht11_f[0] + dht11_f[1] + dht11_f[2] + dht11_f[3] & 0xFF);
89      }
90  }
91
```

설명:

- `EasyDHT11` 클래스는 DHT11 센서의 데이터 읽기와 체크섬 검증을 담당합니다.
- `getData()` 메서드는 센서에서 온도와 습도 데이터를 읽어 반환합니다.

2. 각 구현 기능 설명

- **CoAP 서버**: `ServerCoAP` 클래스는 CoAP 서버를 시작하고, 리소스를 서버에 등록한 후 클라이언트 요청에 응답합니다.
- **DHT11 센서 데이터**: `DHT11` 클래스는 온도와 습도 값을 읽어 CoAP GET 요청에 응답합니다.
- **Relay 제어**: `Relay` 클래스는 릴레이의 상태를 제어하며, 클라이언트 요청에 따라 핀을 설정합니다.
- **GPIO 제어**: `EasyDHT11` 클래스는 Raspberry Pi의 GPIO를 사용하여 DHT11 센서와 릴레이를 제어합니다.

2. Client 소스 코드 설명

2.1 ClientCoAP 클래스

이 코드는 CoAP (Constrained Application Protocol)을 사용하여 스마트 환기 장치와 통신하는 클라이언트를 구현한 예시입니다. CoAP는 IoT 환경에서 자주 사용되는 경량화된 프로토콜로, HTTP보다 낮은 오버헤드로 통신을 할 수 있습니다. 이 클라이언트는 GUI 환경을 제공하며, 다양한 CoAP 요청(GET, POST, PUT, DELETE)을 지원하고, 릴레이 및 DHT11 센서 데이터를 제어하는 기능을 제공합니다.

주요 기능 설명

1. CoAP 요청 처리

- **GET, POST, PUT, DELETE** 버튼을 통해 서버에 요청을 전송합니다.
- 요청은 `CoapRequestCode`를 사용하여 종류를 지정하며, 각 요청에 필요한 경로(`path`)와 페이로드(`payload`)를 입력받습니다.

2. 릴레이 제어

- **Relay On** 및 **Relay Off** 버튼을 통해 릴레이의 상태를 변경할 수 있습니다.
- 릴레이 상태는 `"/relay"` 경로로 PUT 요청을 통해 전송됩니다. `"1"`은 릴레이 켜기, `"0"`은 릴레이 끄기입니다.
- **Relay Info** 버튼은 릴레이의 현재 상태를 조회하는 GET 요청을 보냅니다.

3. DHT11 센서 정보 요청

- **DHT11 Info** 버튼은 DHT11 센서에서 측정된 데이터를 요청합니다.
- 이 요청은 `"/dht11"` 경로를 사용하여 GET 요청을 보냅니다.

4. GUI 구성

- JButton 컴포넌트를 사용하여 각 버튼을 배치하고, 이벤트를 처리합니다.
- JTextArea와 JLabel을 사용하여 경로, 페이로드, 응답 등을 표시합니다.
- JScrollPane을 사용하여 긴 응답 메시지를 스크롤할 수 있도록 처리합니다.

5. 응답 처리

```
193     @Override
194     public void onResponse(CoapClientChannel channel, CoapResponse response) {
195         byte[] payload = response.getPayload();
196         if (payload != null && payload.length > 0) {
197             String responseStr = new String(payload); // 바이트 배열을 문자열로 변환
198             display_text.append("Response: " + responseStr); // 화면에 응답 데이터 표시
199         } else {
200             display_text.append(str:"Response: No payload");
201         }
202         display_text.append(System.lineSeparator());
203     }
```

- 서버에서 응답을 받으면, 응답 데이터를 JTextArea에 출력합니다.

6. CoAP 요청 전송

```
138     /* Method */
139     private void sendRequest(CoapRequestCode requestCode) {
140         String path = path_text.getText();
141         String payload = payload_text.getText();
142 
143         CoapRequest request = clientChannel.createRequest(requestCode, path, reliable:true);
144 
145         // PUT과 POST 요청일 경우, Payload 처리
146         if (requestCode == CoapRequestCode.PUT || requestCode == CoapRequestCode.POST) {
147             byte[] payloadBytes = new byte[] { (byte) (payload.equals(anObject:"1") ? 1 : 0) };
148             request.setPayload(new CoapData(payloadBytes, CoapMediaType.text_plain));
149         }
150 
151         displayRequest(request);
152         clientChannel.sendMessage(request);
153     }
```

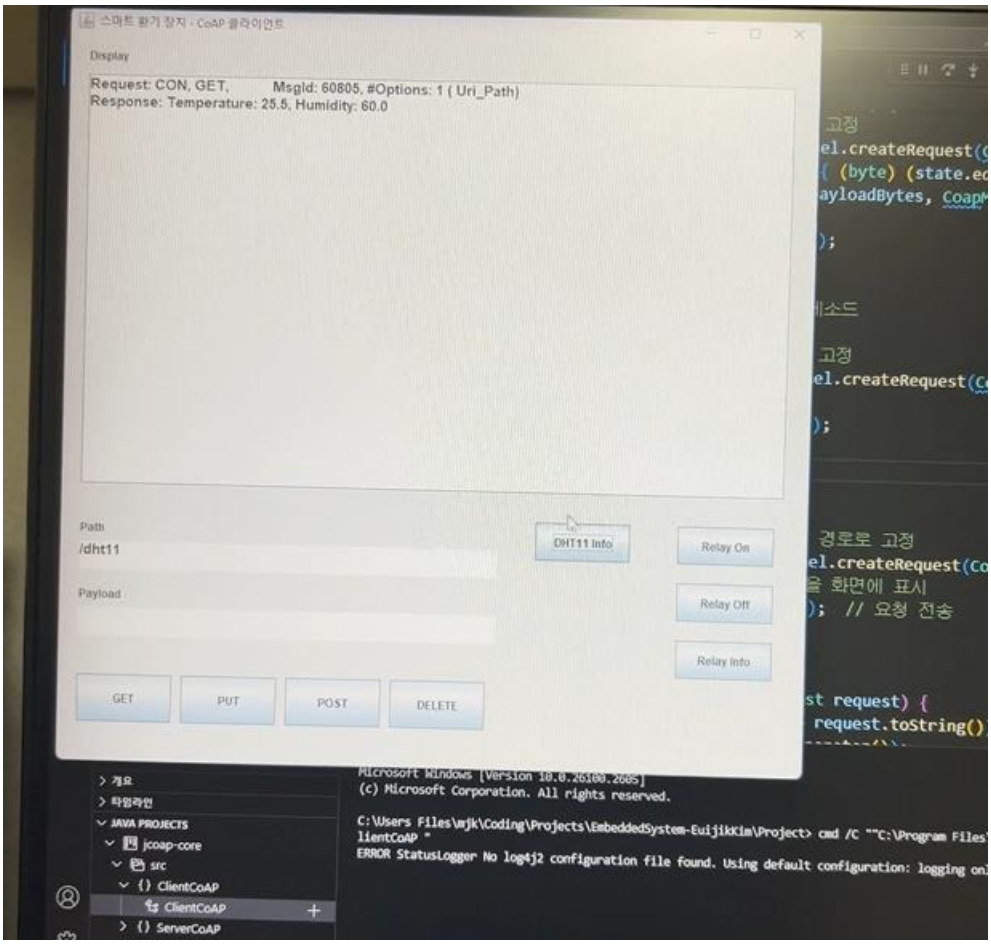
- CoapRequestCode에 따라 GET, POST, PUT, DELETE 요청을 생성하여 서버에 전송합니다.
- PUT과 POST 요청의 경우, 페이로드를 설정하여 전송합니다.

GUI 레이아웃

- **버튼 배치**: 버튼은 `setBounds()` 메서드를 사용하여 위치를 설정합니다. 버튼을 클릭하면 해당하는 요청이 전송됩니다.
- **텍스트 필드**: 경로와 페이로드를 입력하는 `JTextArea`와, 응답을 출력하는 `JTextArea`가 사용됩니다.

3. 실행 화면

- 클라이언트 GUI 에서 [DHT11 Info] 버튼을 눌렀을때



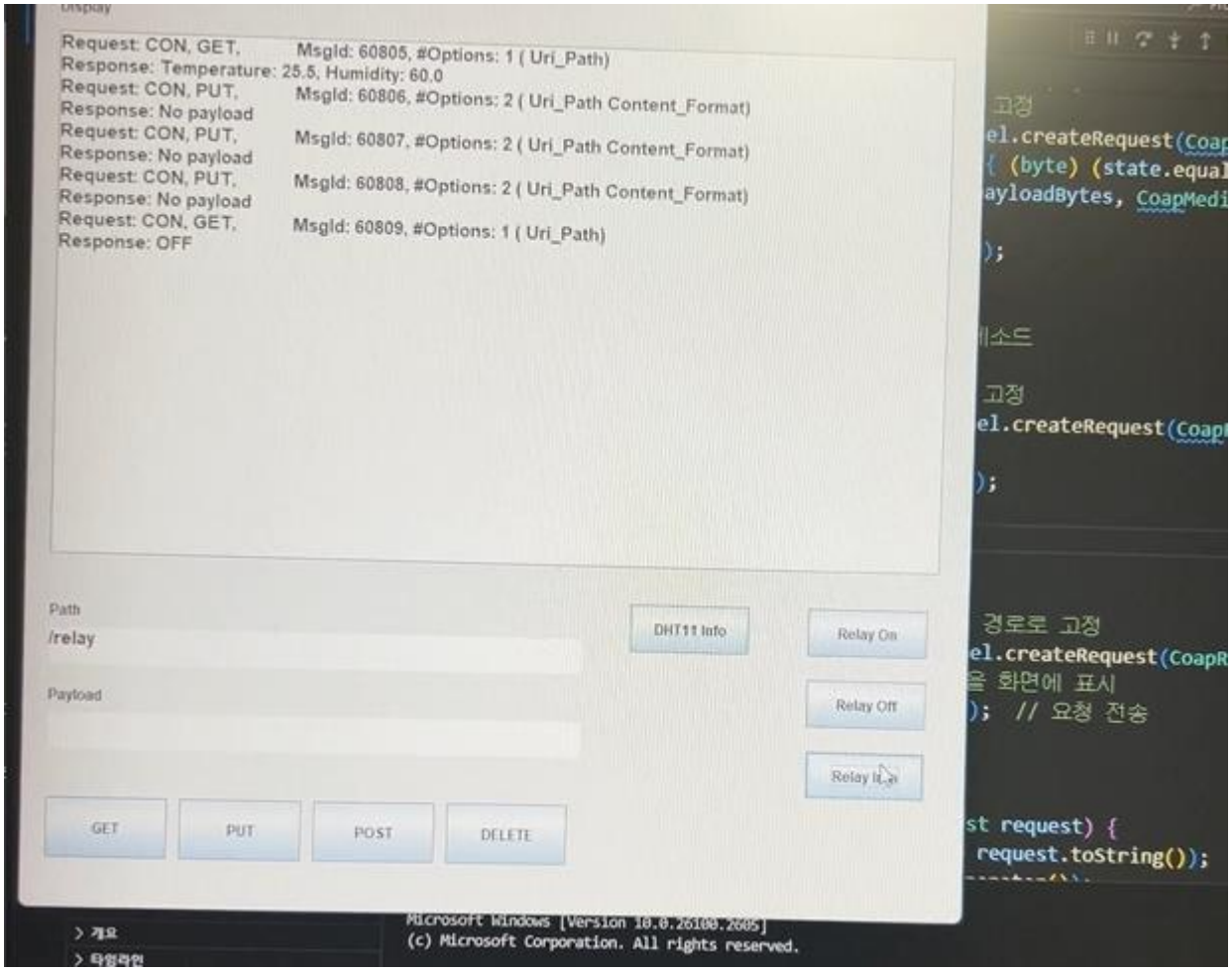
다음과 같이 온도와 습도 값이 나옵니다. 또한 Path, Payload 필드 역시 자동으로

값이 변합니다. 이후 나머지 버튼들도 마찬가지로 버튼을 누르면 편리하게 자동으로 두 필드를 수정합니다.

- 클라이언트 GUI 에서 [Relay On/Off] 버튼을 눌렀을때

다음과 과 같이 환풍기가 켜지고 꺼지는 작동을 수행합니다.

- 클라이언트 GUI 에서 [Relay Info] 버튼을 눌렀을때



다음과 같이 현제 환풍기 팬의 상태를 알려줍니다 OFF == 꺼져있음, ON == 켜져있음

III. 결론

이번 프로젝트에서는 스마트 환기 장치를 제어하는 CoAP 클라이언트의 구현 과정을 상세히 다루었습니다. 구현된 CoAP 클라이언트는 다양한 기능을 통해 서버와의 통신을 가능하게 하며, CoAP 프로토콜을 사용하여 IoT 장치와의 상호작용을 처리합니다. 주요 기능으로는 GET, POST, PUT, DELETE 요청을 통한 서버와의 데이터 통신, 릴레이 제어, 그리고 DHT11 센서를 통해 온도 및 습도 정보를 요청하는 기능을 포함합니다.

각 버튼을 클릭하여 다양한 기능을 손쉽게 실행할 수 있도록 GUI 환경을 제공하며, 사용자가 실시간으로 요청 및 응답을 확인할 수 있게 하여 시스템 상태를 직관적으로 모니터링할 수 있

습니다. 클라이언트는 `CoapClient` 인터페이스를 구현하고, `CoapChannelManager`를 사용하여 서버와의 연결을 관리하며, 각 요청은 적절한 경로와 페이로드를 설정해 서버로 전송됩니다.

본 구현은 IoT 장치의 제어 및 데이터를 관리하는 데 유용한 예시를 제공하며, CoAP 프로토콜을 활용하여 네트워크 환경에서의 효율적인 데이터 전송 및 장치 제어를 가능하게 합니다. 이와 같은 시스템을 통해 스마트 환기 장치와 같은 IoT 기기의 제어가 용이해지며, 더욱 효과적인 환경 제어 및 모니터링이 이루어질 수 있습니다.

종합적으로, 본 CoAP 클라이언트는 스마트 홈 또는 IoT 환경에서 다양한 장치와의 연결 및 관리 기능을 효율적으로 처리하는 중요한 역할을 하며, 향후 더 다양한 IoT 장치와의 통신을 확장할 수 있는 기반을 마련합니다.

<참고자료>

1. *없음.*