

온습도 변화 시각화 LED 모니터링 시스템

박혜진, 20235177, 스마트 IoT

목차

프로젝트 제목.....	1
이름, 학번, 학과.....	1
I. 서론.....	2
1. 프로젝트의 필요성.....	2
2. 프로젝트의 목적.....	2
II. 본론.....	3
1. 하드웨어 구현.....	3
2. 소프트웨어 구현.....	4
III. 결론.....	5
<참고자료>.....	6



한림대학교
HALLYM UNIVERSITY

I. 서론

1. 프로젝트의 필요성

더 웰스 : 건강한 가족

"꿀잠 좀 자고 싶다"...이런 분들 '방온도'부터 체크를 [건강한 가족]

중앙일보 | 입력 2024.12.09 05:31 | [자면보기](#) ①

학자수 기자 **주목**

겨울철 불면증 극복하기

취침 4~5시간 전 커피 피해야
낮잠 15분 내로, 야밤 운동은 '독'
우유·호두·바나나 등 수면에 도움



겨울이면 불면증을 호소하는 사람이 늘어난다. 떨어진 낮 기온으로 생체리듬이 깨지고 건조한 공기 탓에 수면무호흡증 같은 수면 장애가 심해지는 영향이다. 창문은 닫아 놓고 실내를 따뜻하게 유지하는 게 중요하다. 하루 동안 쌓인 피로를 풀고 에너지를 충전하지 못해 집중력도 떨어진다는 지적이다. 면역력이 떨어져 각종 질환에 노출될 가능성도 커진다. 가나안 겨울철, 불면증을 개선하고 일상에 활기를 더할 생활지침을 살펴본다.

눈 건조할 때... 습도보다 '이것' 올려야

이슬비 기자 | 입력 2024.12.12 06:00



클립아트코리아

차고 건조한 바람이 부는 겨울이면 많은 사람이 안구 건조증을 호소한다. 건강보험심사평가원에서 발표한 지난해 월별 안구건조증 환자 수를 보면, 찬바람이 불기 직전인 9월에는 30만 6113명이었지만 12월에는 34만 9695명으로 늘었다. 보통 대기가 건조해져서라고 생각하는데, 실은 기온이 미치는 영향이 더 크다. 눈이 건조할 때 습도보다 온도를 높이면 더 효과적으로 증상을 개선할 수 있다.

안구 건조증은 눈 표면에 있어야 할 눈물 양이 부족해 생기는 염증 질환으로, 방치하면 눈이 뻑뻑하고 모래알이 들어간 것처럼 불편하다. 안구 보호 기능이 떨어져 낮은 자극에도 쉽게 눈물이 흐르고, 장시간 전자기기를 보며 자극을 가하면 눈꺼풀이 달라붙거나 충혈된다. 악화하면 만성질환으로 발전해 결막염, 각막염, 시력저하 등 합병증을 유발할 수도 있다. 일상생활의 질을 크게 떨어뜨리는 질환이라, 적기에 치료 받는 게 매우 중요하다.

기존의 온도 및 습도 모니터링 시스템은 대개 정적인 데이터 표시만을 제공하거나, 온도 및 습도를 설정된 범위 내에 유지하는 기능에 집중하는 경향이 있다. 그러나 실내 환경의 변화를 실시간으로 시각적으로 확인할 수 있는 기능이 부족해, 사용자들이 온도와 습도 변화에 즉각적으로 반응하지 못하는 경우가 많다. 특히 숙면과 안구 건강을 위해 적정 온도와 습도 조절이 중요한 겨울철에는 이러한 실시간 모니터링의 중요성이 더욱 커진다. 이를 해결하기 위해 제안하는 온도 및 습도에 따른 LED 색상 변화를 통해 사용자는 실내 환경의 변화를 직관적으로 인식하고, 보다 효율적으로 환경을 조절할 수 있다. 이 프로젝트는 사용자가 건강한 실내 환경을 유지하고, 숙면과 안구 건조증 예방에 도움이 되는 최적의 온도와 습도를 쉽게 관리할 수 있도록 돕는 중요한 역할을 한다.

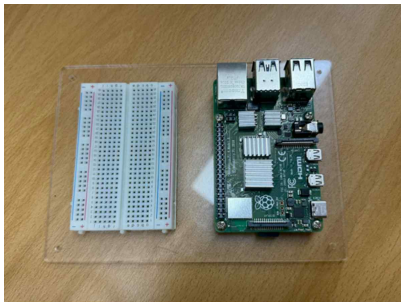
2. 프로젝트의 목적

DHT11 센서를 활용하여 실내 온도와 습도를 실시간으로 모니터링하고, LED 색상 변화를 통해 사용자에게 직관적인 환경 정보를 제공함으로써 사람들이 쉽고 빠르게 실내 환경을 관리하고 조절할 수 있도록 돕는 것이다. 궁극적으로는 건강한 실내 환경 조성을 통해 수면의 질 개선과 안구 건강 보호를 목표로 한다.

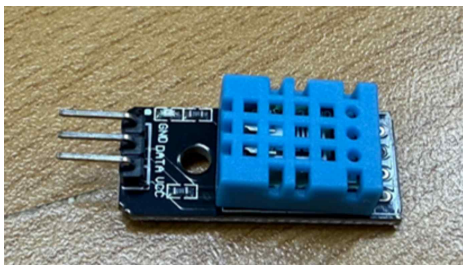
II. 본론

1. 하드웨어 구현

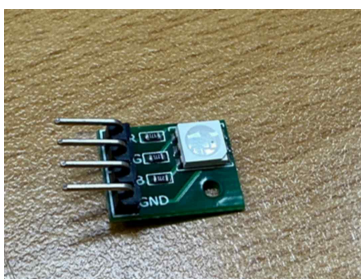
Raspberry Pi: Raspberry Pi 4 모델을 사용하여 시스템의 제어와 데이터 처리 작업을 수행한다. Raspberry Pi는 GPIO 핀을 이용해 센서 및 액추에이터와 연결되며, 데이터 수집, 처리 및 전송을 담당한다.



DHT11 온습도 센서: 온도와 습도를 측정해 Raspberry Pi에 데이터를 전달하는 역할을 한다. DHT11 센서는 디지털 신호를 통해 데이터를 전송하며, 이것을 통해 실시간 온도와 습도 값을 정확하게 측정할 수 있다.

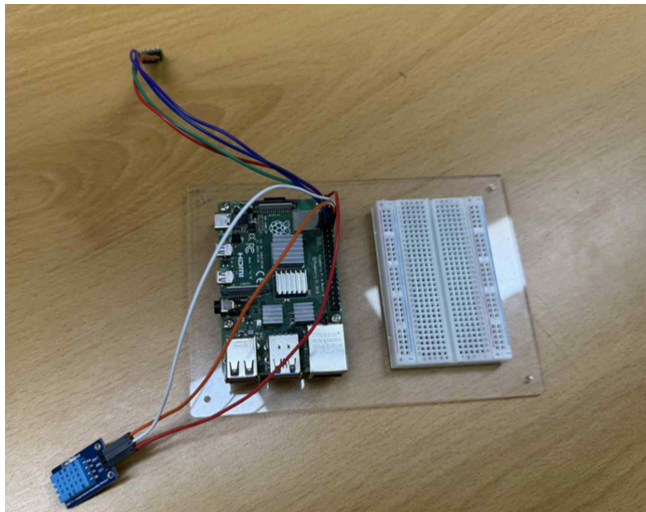


RGB LED 센서: 온도 및 습도 값에 따라 색상을 변경하는 액추에이터로 사용된다. 온도가 30도 이상일 때 빨간색, 습도가 80% 이상일 때 파란색, 온도가 0도 이하일 때 흰색, 그 외에는 초록색으로 표시되도록 설정하여, 사용자에게 직관적으로 환경 변화를 알릴 수 있다. RGB LED의 각 색상은 Raspberry Pi의 GPIO 핀을 통해 제어된다.



2024년도 2학기 – 임베디드시스템 프로젝트 보고서

GPIO#	NAME		NAME	GPIO#
	3.3 VDC Power	1	2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3	4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5	6	Ground
7	GPIO 7 GPCLK0	7	8	GPIO 15 TxD (UART) 15
	Ground	9	10	GPIO 16 RxD (UART) 16
0	GPIO 0	11	12	GPIO 1 PCM_CLK/PWM0 1
2	GPIO 2	13	14	Ground
3	GPIO 3	15	16	GPIO 4 4



[DHT11 센서]

DHT11 온습도 센서	GPIO Pins
DOUT	15
GND	Ground
VCC	3.3 VDC

[RGB LED 센서]

LED 센서	GPIO Pins
R	8
G	9
B	7
VCC	Ground

2. 소프트웨어 구현

1. Sever 소스 코드 설명

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

public class CoAP {
    private static CoAP coapServer;
    private CoapResourceServer resourceServer;

    public static void main(String[] args) {
        coapServer = new CoAP();
        coapServer.start();
    }

    public void start() {
        System.out.println("===Run Test Server ===");

        // create server
        if (this.resourceServer != null)
            this.resourceServer.stop();
        this.resourceServer = new CoapResourceServer();

        // initialize resource
        LED led = new LED();
        DHT dht = new DHT();

        //CoapResourceServer에 observe하려는 resource 등록
        dht.registerServerListener(resourceServer);

        // add resource to server
        this.resourceServer.createResource(dht);
        this.resourceServer.createResource(led);

        // run the server
        try {
            this.resourceServer.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
        while (true) {
            try {
                Thread.sleep(5000);
                dht.changed();
                led.changed();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

- CoapResourceServer 객체를 생성하여 CoAP 서버를 초기화한다. 이미 서버가 실행 중이라면 기존 서버를 중지하고 새로 시작한다.
- LED와 DHT라는 리소스 객체를 생성한다.
- DHT 리소스를 관찰 가능하게 설정하기 위해 dht.registerServerListener(resourceServer)를 호출한다.
- this.resourceServer.createResource()를 호출하여 DHT와 LED 리소스를 서버에 추가한다.
- 서버를 시작하며, 무한 루프를 통해 5초 간격으로 dht.changed()와 led.changed()를 호출하여 리소스 상태가 변경되었음을 client에게 알린다. 이는 CoAP의 Observe 기능을 통해 상태 변화를 클라이언트로 전달한다.

2. Client 소스 코드 설명

```
// 색상 정의
Color backgroundColor = new Color(214, 230, 245); // 전체 배경 (파스텔 블루)
Color displayBackgroundColor = new Color(255, 250, 240); // display_text 배경 (크림색)
Color textColor = new Color(85, 58, 48); // 일반 텍스트 (다크 브라운)
Color buttonBackgroundColor = new Color(255, 250, 240); // 버튼 배경 (크림색)
Color buttonTextColor = new Color(85, 58, 48); // 버튼 텍스트 (다크 브라운)

// 프레임 배경색 설정
this.getContentPane().setBackground(backgroundColor);

// display_text 색상 설정
display_text.setBackground(displayBackgroundColor);
display_text.setForeground(textColor);

// 버튼 색상 설정
btn_get.setBackground(buttonBackgroundColor);
btn_get.setForeground(buttonTextColor);
btn_observe.setBackground(buttonBackgroundColor);
btn_observe.setForeground(buttonTextColor);
btn_put.setBackground(buttonBackgroundColor);
btn_put.setForeground(buttonTextColor);
btn_delete.setBackground(buttonBackgroundColor);
btn_delete.setForeground(buttonTextColor);

// 텍스트 필드 설정
payload_text.setBackground(displayBackgroundColor);
payload_text.setForeground(textColor);
path_text.setBackground(displayBackgroundColor);
path_text.setForeground(textColor);

// 라벨 색상 설정
path_label.setForeground(textColor);
payload_label.setForeground(textColor);
display_label.setForeground(textColor);

// Button setup
btn_get.setBounds(20, 670, 150, 50);
btn_observe.setBounds(180, 670, 150, 50);
btn_put.setBounds(340, 670, 150, 50);
btn_delete.setBounds(600, 670, 150, 50);
```

- GUI창 간격 설정 및 색상 설정

```
// GET button action listener
btn_get.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (deleteCalled) return;
        String path = path_text.getText();
        String payload = payload_text.getText();
        CoapRequest request = clientChannel.createRequest(CoapRequestCode.GET, path, true);
        displayRequest(request);
        clientChannel.sendMessage(request);
    }
});
```

- GET 버튼을 클릭하면, 현재 입력된 경로(path_text)와 페이로드(payload_text) 값을 읽고, CoapRequestCode.GET 요청을 생성하여 서버로 전송한다. 만약 DELETE가 이미 호출되었다면 요청을 보내지 않고 종료된다. clientChannel.sendMessage(request)로 전송되며, 요청이 화면에 표시된다.

```
// PUT button action listener
btn_put.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (deleteCalled) return;
        String path = path_text.getText();
        String payload = payload_text.getText();
        CoapRequest request = clientChannel.createRequest(CoapRequestCode.PUT, path, true);
        request.setPayload(new CoapData(payload, CoapMediaType.text_plain));
        displayRequest(request);
        clientChannel.sendMessage(request);
    }
});
```

- PUT 버튼을 클릭하면, 입력된 경로와 페이로드를 기반으로 CoapRequestCode.PUT 요청을 생성한다. 요청에 입력된 페이로드(payload_text)를 CoapData로 변환하여 추가하고, 해당 요청을 서버로 전송한다. DELETE가 이미 호출된 경우, 요청은 전송되지 않는다.


```
// DELETE button action listener (Stop observing and delete the resource)
btn_delete.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        deleteCalled = true;
        String path = path_text.getText();
        CoapRequest request = clientChannel.createRequest(CoapRequestCode.DELETE, path, true);
        displayRequest(request);
        clientChannel.sendMessage(request);

        // Stop observation when DELETE is called
        if (observing) {
            stopObservation();
        }
    }
});
```

- DELETE 버튼을 클릭하면, deleteCalled 플래그를 true로 설정하여 다른 요청이 실행되지 않도록 한다. 그 후, 지정된 경로에 대해 CoapRequestCode.DELETE 요청을 생성하여 서버로 전송한다. 현재 리소스를 관찰 중인 경우에는 stopObservation()을 호출하여 관찰을 중지한다.

```
// Observe button action listener
btn_observe.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (deleteCalled) return;
        String path = path_text.getText();
        String payload = payload_text.getText();
        if (observing) {
            stopObservation();
        } else {
            startObservation(path);
        }
    }
});
```

- OBSERVE 버튼을 클릭하면, DELETE가 호출된 상태가 아니면 동작을 시작한다. 만약 현재 관찰 중이라면, stopObservation() 메서드를 호출하여 관찰을 중지하고, 그렇지 않으면 startObservation(path) 메서드를 호출하여 새로운 관찰을 시작한다.

```
@Override
public void onResponse(CoapClientChannel channel, CoapResponse response) {
    if (deleteCalled) {
        System.exit(0);
        return;
    }

    if (response.getPayload() != null) {
        String payload = Encoder.ByteString(response.getPayload());

        try {
            String[] parts = payload.split(",");
            if (parts.length == 2) {
                try {
                    double temperature = Double.parseDouble(parts[0].trim());
                    double humidity = Double.parseDouble(parts[1].trim());
                    String ledColor = decideLedColor(temperature, humidity);
                    display_text.append("Temperature: " + temperature + "°C, Humidity: " + humidity + "%\n");
                    display_text.append("LED Color: " + ledColor + "\n");
                } catch (NumberFormatException e) {
                    display_text.append("Error: Invalid number format in payload.\n");
                }
            } else {
                display_text.append("Error: Payload format - 'Temperature,Humidity'.\n");
            }
        } catch (Exception e) {
            display_text.append("Error: " + e.getMessage() + "\n");
        }
    } else {
        // Handle if there's no payload
        display_text.append(System.lineSeparator());
        display_text.append("");
        display_text.append(System.lineSeparator());
    }
}
```

- CoAP 클라이언트에서 서버로부터 받은 응답을 처리하는 메서드로, 먼저 deleteCalled가 true 인 경우 프로그램을 종료하고, 그렇지 않으면 응답의 페이로드가 존재하는지 확인한다. 페이로드가 있으면 이를 문자열로 변환한 뒤, 쉼표로 구분하여 온도와 습도 데이터를 추출하며,

데이터가 정확히 두 부분(온도와 습도)으로 나뉘어 있으면 이를 double로 변환하고, decideLedColor 메서드를 통해 온도와 습도 값에 따라 LED 색상을 결정한다. 변환된 데이터와 LED 색상은 display_text에 추가해 사용자에게 표시되며, 데이터 형식이 잘못되었거나 숫자 변환에 실패한 경우 적절한 오류 메시지를 출력한다. 마지막으로 응답 처리가 끝난 후 구분선(*)을 출력하여 결과를 정리한다.

```
private String decideLedColor(double temperature, double humidity) {
    if (temperature == -99.0 || humidity == -99.0) {
        return "off";
    } else if (temperature <= 0) {
        return "White";
    } else if (temperature >= 30) {
        return "Red";
    } else if (humidity >= 80) {
        return "Blue";
    } else {
        return "Green";
    }
}
```

- 온도와 습도를 입력받아 조건에 따라 LED 색상을 결정한다. 온도나 습도가 -99.0인 경우 센서가 비활성 상태임을 의미하며 LED는 꺼짐(off) 상태로 설정된다. 온도가 0°C 이하일 때는 LED를 흰색(White)으로, 30°C 이상일 때는 빨간색(Red)으로 설정된다. 습도가 80% 이상이면 LED는 파란색(Blue)으로 설정되고, 나머지 경우에는 녹색(Green)으로 설정된다. 이 방식으로 센서 데이터를 기반으로 LED의 상태를 직관적으로 나타낸다.

```
private void displayRequest(CoapRequest request) {
    if (request.getPayload() != null) {
        display_text.append("Path: " + request.getUriPath() + "\n");
        display_text.append("Payload: " + Encoder.ByteString(request.getPayload()) + "\n");
    }
    display_text.append(System.LineSeparator());
    display_text.append("*");
    display_text.append(System.LineSeparator());
}
```

- CoAP 요청을 화면에 출력하는 기능을 담당합니다. 요청에 페이로드가 포함되어 있다면, 요청의 URI 경로와 페이로드 데이터를 문자열로 변환하여 display_text에 추가한다. 이후 구분선(*)을 출력하여 요청 내용을 구분한다.

```
// Start observing the resource
private void startObservation(String path) {
    CoapRequest request = clientChannel.createRequest(CoapRequestCode.GET, path, true);
    request.setObserveOption(0); // Start observation
    displayRequest(request);
    clientChannel.sendMessage(request);
    observing = true;
}

// Stop observing the resource
private void stopObservation() {
    CoapRequest request = clientChannel.createRequest(CoapRequestCode.GET, path_text.getText(), true);
    request.setObserveOption(-1); // Cancel observation
    displayRequest(request);
    clientChannel.sendMessage(request);
    observing = false;
}
```

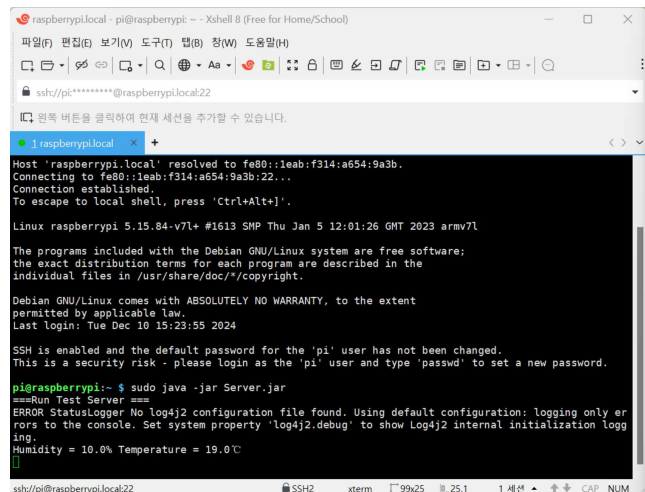
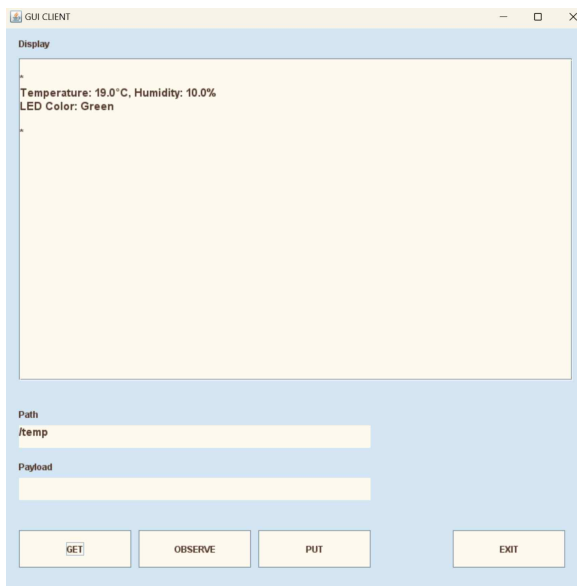
- startObservation 메서드) CoAP 클라이언트가 지정된 경로(path)의 리소스를 관찰하기 시작하도록 설정한다. 관찰을 시작하려면 GET 요청을 생성하고, Observe 옵션 값을 0으로 설정하여 서버에 요청한다. 생성된 요청은 displayRequest 메서드로 화면에 출력되며, 이후

clientChannel.sendMessage를 호출해 서버로 전송된다. 관찰 상태를 나타내는 observing 플래그는 true로 설정된다.

- stopObservation 메서드) 클라이언트가 리소스 관찰을 중지하도록 한다. 현재 입력된 경로를 사용해 GET 요청을 생성하며, Observe 옵션 값을 -1로 설정하여 서버에 관찰 중지 요청을 보낸다. 요청은 displayRequest 메서드를 통해 화면에 출력되고, clientChannel.sendMessage를 통해 서버로 전송된다. 관찰 상태를 나타내는 observing 플래그는 false로 설정된다.

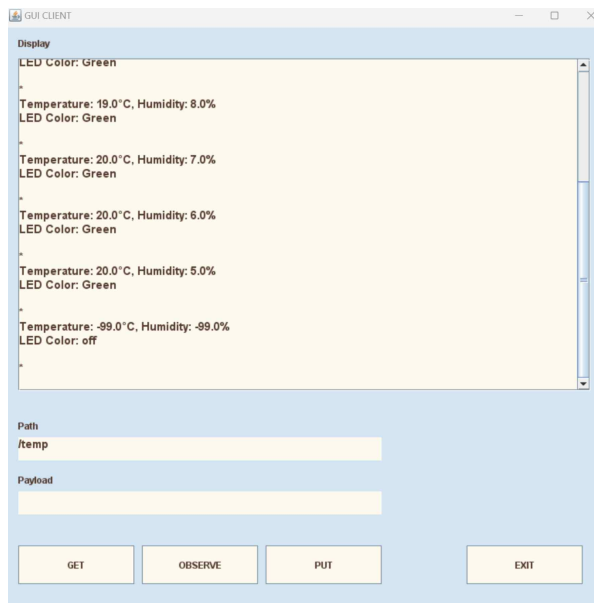
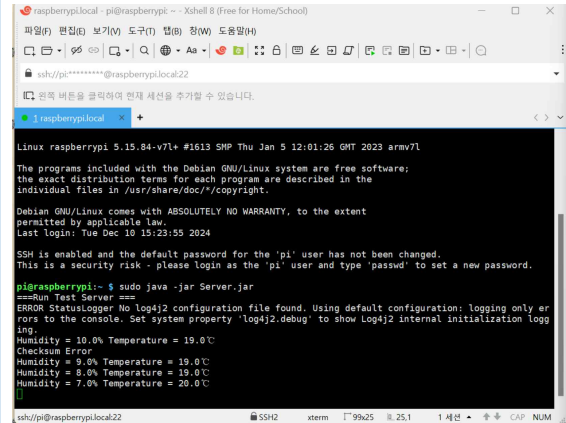
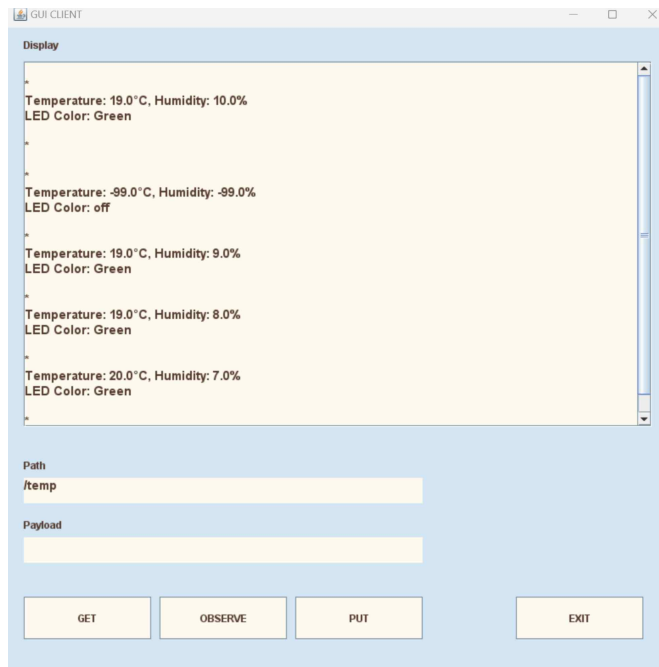
3. 실행 화면

GET 요청)



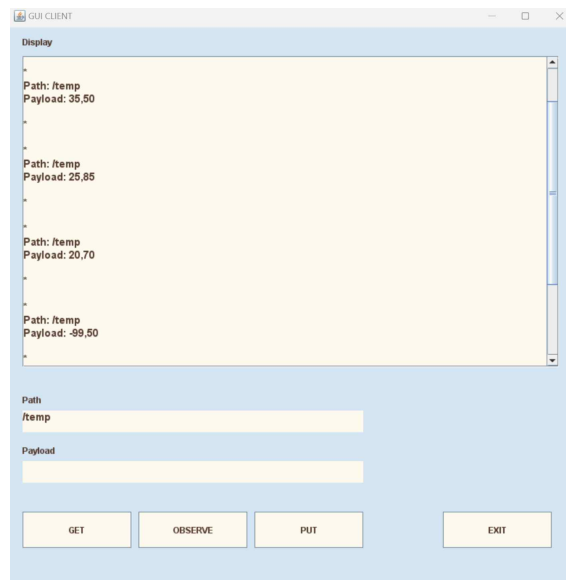
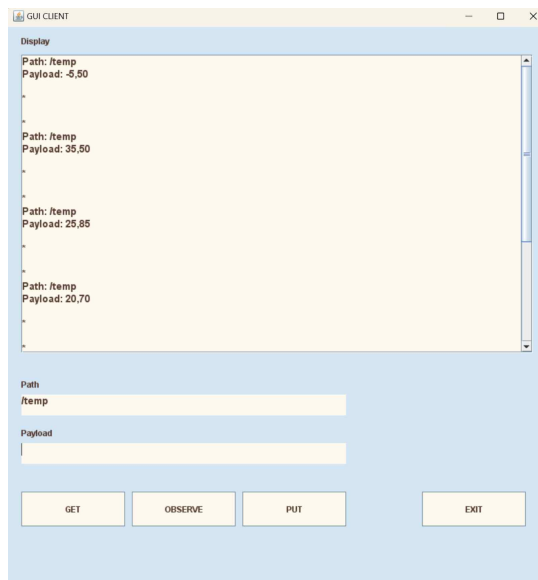
2024년도 2학기 - 임베디드시스템 프로젝트 보고서

OBSERVE 요청)

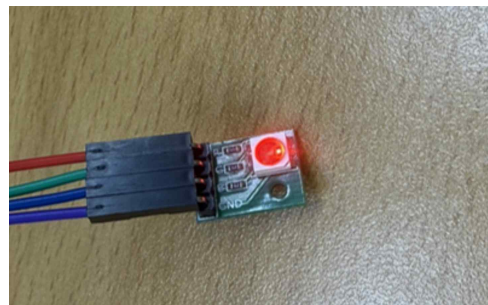


- Checksum 에러가 나올 시, led off

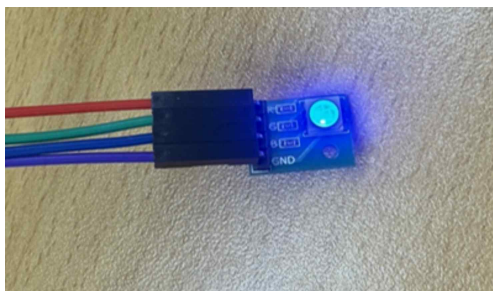
PUT 요청)



- Payload 값: -5,50 (흰색)



- Payload 값: 35,50 (빨간색)



- Payload 값 : 25, 85 (파란색)

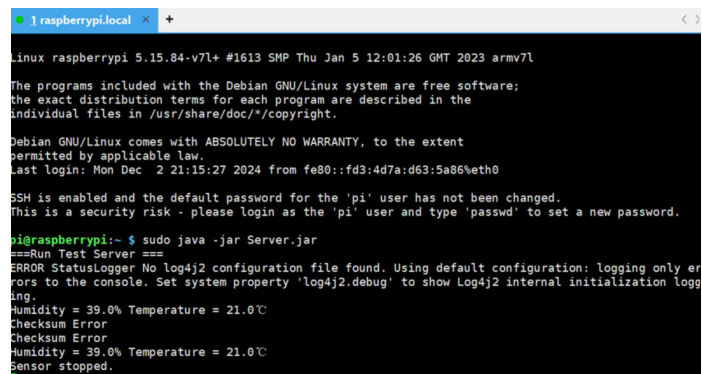


- Payload 값 : 20,70 (초록색)



- Payload 값 : -99,50 (off)

EXIT 호출)



```
linux raspberrypi 5.15.84-v7l+ #1613 SMP Thu Jan 5 12:01:26 GMT 2023 armv7l
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Dec  2 21:15:27 2024 from fe80::fd3:4d7a:d63:5a86%eth0
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.
pi@raspberrypi:~$ sudo java -jar Server.jar
===Run Test Server===
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only er
rors to the console. Set system property 'log4j2.debug' to show Log4j2 internal initialization logg
ing.
Humidity = 39.0% Temperature = 21.0℃
Checksum Error
Checksum Error
Humidity = 39.0% Temperature = 21.0℃
Sensor stopped.
```

- GUI 창이 닫히며, sensor도 작동을 멈춘다.

III. 결론

본 프로젝트에서는 CoAP 프로토콜을 활용하여 온도 및 습도 센서(DHT11)와 LED 조명을 제어하는 시스템을 구현하였다. 센서에서 수집된 온도와 습도 데이터는 서버와 클라이언트 간의 CoAP 메시지를 통해 주고받으며, 이를 바탕으로 LED 색상이 변경된다. 클라이언트 GUI는 GET, PUT, DELETE, OBSERVE와 같은 CoAP 메서드를 통해 서버와 상호작용하며, 실시간으로 센서 데이터를 확인하고 제어할 수 있다. 이를 통해 IoT 환경에서의 실시간 데이터 처리 기능을 실현하였다.

PUT 요청은 시스템의 핵심 동작에는 영향을 미치지 않으며, 주로 테스트와 검증을 위한 용도로 사용되었다. 예를 들어, LED 상태를 변경하는 요청을 통해 하드웨어와의 상호작용을 시뮬레이션하고, 시스템의 응답을 점검하는 방식으로 활용되었다. 따라서 실제 운영 시스템에서는 중요한 기능이 아니며, 기능 구현 및 검증 과정에서만 존재한 요청이다.