8조 IR HUB 결과보고서

개발 목표

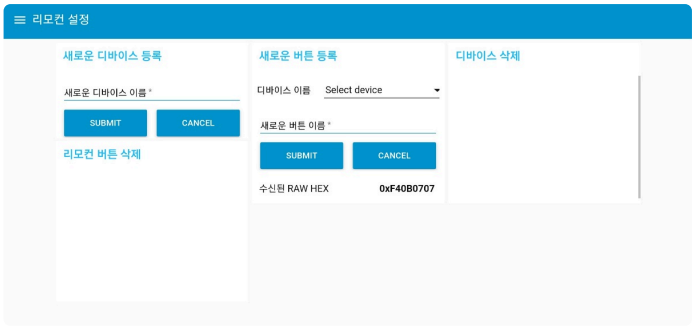
프로젝트 목표

실내에서 디바이스를 무선으로 제어하기 위해 가장 흔하게 사용되는 무선 통신 방법인 적외선 통신을 사용하여 IR HUB를 제작하는 것이 이번 프로젝트의 목표이다.

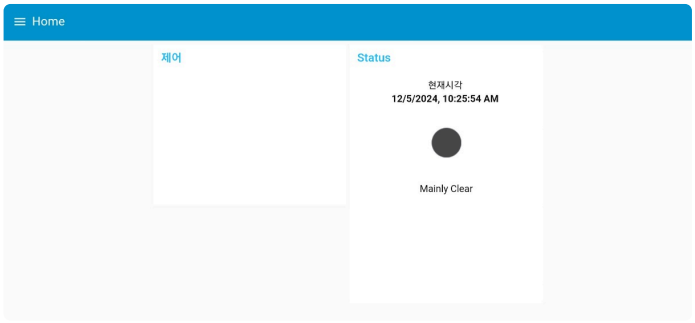
- IoT가 지원되지 않는 기존 디바이스들도 IoT를 이용하여 제어할 수 있다는 장점이 있다.
- 외부에서도 집 내부의 디바이스 제어가 가능하고, 실내 디바이스를 자동으로 제어할 수 있다.

최종 개발 결과물

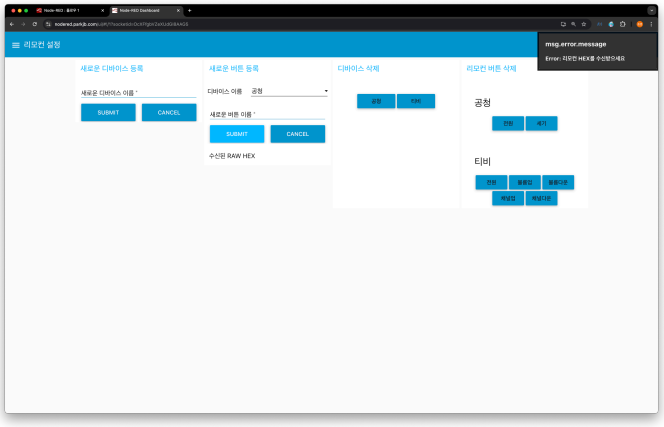
리모컨 설정 초기 화면



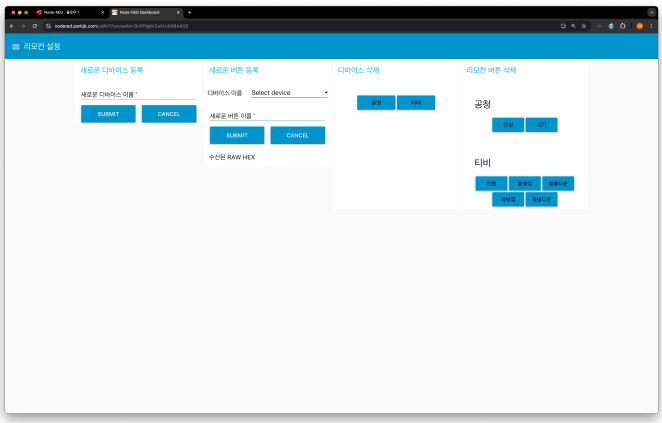
제어 초기 화면



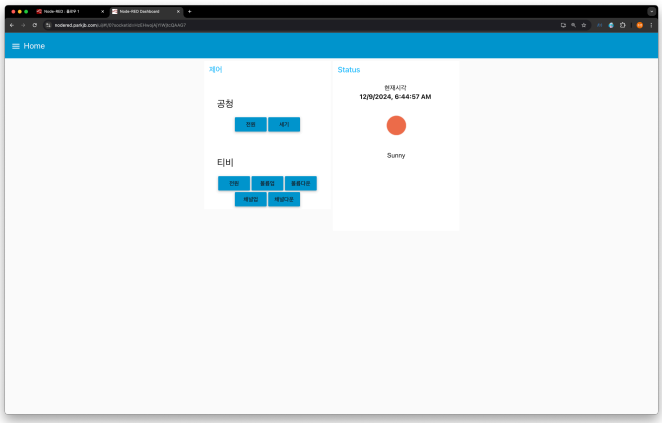
에러메시지 표시



IR 등록 후 리모컨 설정 화면



IR 등록 후 제어 화면

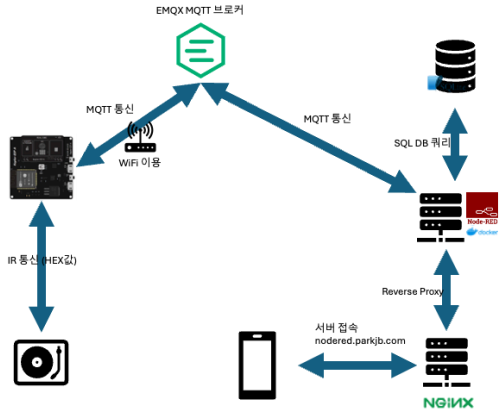


팀 구성

- 박종범: 아이디어 구상, Node-RED 개발, 서버 개발 및 아키텍쳐 구성
- 이한빈: 아이디어 구상, Kepler-ESP A 개발

개발 방법

개발 기술 개요



전체적인 서비스의 흐름도이다.

엔드 유저 (스마트폰)에서 nodered.parkjb.com/ui 에 접속하여 IR Hub의 대시보드에 접근하면, 대시보드 클라이언트에서 새로운 디바이스/버튼을 등록/삭제하는 과정을 수행할 수 있다.

새로운 리모컨 버튼을 등록하고 해당 버튼을 누른다면 NodeRed 서버에서 MQTT를 거쳐 ESP에 IR을 동작시키는 명령을 보내게 되고, 이를 수행할 경우 최종적으로 홈 안의 가전이 제어되는 방식이다.

사용된 주요 기술

Kepler ESP A 보드

- MQTT를 통해 수신 받은 값에 맞는 동작을 진행
- 적외선을 수신하라는 동작을 입력 받으면 적외선 수신기를 이용하여 적외선을 수신 받고 그 값을 저장
- 적외선 송신기: Kepler-ESP A에서 동작을 하기 위해 적외선을 송신하라는 입력을 받으면 그에 맞는 적외선 값을 송신
- 적외선 수신기: Kepler-ESP A에서 동작에 맞는 적외선 값을 수신하라는 입력을 받으면 적외선을 수신하고 그 적외선 값이 어떠한 동작인지 저장한다.

WiFi & MQTT

WiFi와 MQTT 프로토콜을 이용하여 Node-Red 서버와 통신할 수 있도록 하였다.
MQTT의 브로커로 EMQX public broker를 이용하였으며, 크게 `topic/register_hex`와 `topic/control` 토픽을 이용하였다.

적외선 통신

적외선 통신을 이용하여 적외선을 수신하면 수신된 데이터는 `IrReceiver.decodedIRData`에 `IRData` 타입으로 저장된다. 이때, `IRData` 타입은 다음과 같이 정의되어 있다.

```
struct IRData {
    decode_type_t protocol;     // UNKNOWN, NEC, SONY, RC5, PULSE_DISTANCE, ...
    uint16_t address;           // Decoded address
    uint16_t command;           // Decoded command
    uint16_t extra;             // Used for Kaseikyo unknown vendor ID. Ticks used for decoding Distance protocol.
    uint16_t numberOfBits;      // Number of bits received for data (address + command + parity) - to determine protocol length if different length are possible.
    uint8_t flags;              // IRDATA_FLAGS_IS_REPEAT, IRDATA_FLAGS_WAS_OVERFLOW etc. See IRDATA_FLAGS_* definitions
    IRRawDataType decodedRawData;    // Up to 32 (64 bit for 32 bit CPU architectures) bit decoded raw data, used for sendRaw functions.
    uint32_t decodedRawDataArray[RAW_DATA_ARRAY_SIZE]; // 32 bit decoded raw data, to be used for send function.
    irparams_struct *rawDataPtr; // Pointer of the raw timing data to be decoded. Mainly the data buffer filled by receiving ISR.
};
```

적외선 데이터가 수신된다면, loop 내의 `IrReceiver.decode()`에서 참을 반환하여 `IrReceiver.decodedIRData`에 IRData 타입의 디코딩된 IR 데이터가 저장된다.
적외선 데이터를 송신한다면, `IrSender.write(IRData* irdata)` 함수를 이용하여 원하는 IR 데이터를 송신할 수 있다.

SQLite

- SQLite는 주로 간단한 데이터베이스를 제작하기 위해 사용되는 DBMS이다.
- 특징적인 점으로, 파일을 데이터베이스로 사용한다.
- 이번 프로젝트에서는 대규모 트래픽이 요구되는 프로젝트가 아닌, 주로 적은 수의 홈 사용자들이 이용하는 서비스이기 때문에 DB를 구축.이용하기에 가벼운 SQLite를 사용하였다.

기술 개발 방향

1. Kepler-ESP A와 적외선 수신기를 이용하여 기존 리모컨의 적외선 값 수신
2. 수신된 적외선 값을 서버(Node-RED)로 전송 및 저장 (MQTT 이용)
3. 스마트폰에서 제어가 가능하도록 Node-RED UI 설정
4. 스마트폰에서 Node-RED의 UI를 이용해 어떤 동작을 실행시킬지 결정
5. 동작에 알맞은 적외선 값을 Kepler-ESP A로 송신
6. Node-RED로부터 수신 받은 적외선 값을 Kepler-ESP A와 적외선 송신기를 이용하여 디바이스에 송신
7. 디바이스에서 수신 받은 적외선 값에 알맞은 동작 진행

기술 개발의 독창성 및 도전성

다양한 IoT 미지원 디바이스의 제어

대부분의 리모컨에 사용되는 적외선 통신을 이용하여 IoT가 지원되지 않는 디바이스더라도 원격으로 디바이스를 제어할 수 있다는 강점이 존재한다.

또, 이를 지원하기 위해서 제어하고 싶은 디바이스에 개별적으로 설치하는 방식이 아닌 IR Hub 하나만 두는 방식으로 비용적인 측면에서도 절감할 수 있으며, 시중에 출시되어있는 유사한 제품들에 비해서도 ESP보드를 활용하여 훨씬 저렴한 가격으로 제작할 수 있다.

DB 관리

사용자가 새로운 디바이스/버튼을 추가하는 과정이 존재하기 때문에, 이를 데이터베이스화 시켜 관리하여야 한다.

처음에는, 이를 관리하기 위해서 단순 파일에 JSON 형태로 새로운 데이터를 추가하고 삭제하는 방법으로 이를 구현하려 하였으나, 직접 데이터를 추가/삭제하는 코드를 구현하게 되면, 데이터의 관리가 복잡해진다는 측면이 존재하여 RDBMS를 도입하여 관리하였다.

동적 UI 생성

사용자가 추가한 디바이스, 버튼에 따라 Node Red의 Dashboard에서 동적으로 UI를 생성하는 것에 대해서 어려움이 존재하였다.

이를 해결하기 위해서, 동적으로 UI를 생성하는 UI Template을 직접 제작하여 사용하였다.

IR Data타입의 복잡성

IR 데이터를 수신할 때, 단순히 이론적인 HEX값만 존재하는 것이 아닌, 프로토콜 타입, 주소, 명령, 플래그 등 다양한 데이터가 존재한다는 것을 확인하였고, 이를 올바르게 구현하기 위해 IR 데이터를 JSON을 이용하여 저장하였다.

개발 내용

MQTT Topic

~~Topic: `topic/rcv_mode`~~

- 리모컨 적외선 값을 수신받는 모드를 켜거나 끄는 토픽
- 항상 수신받을수 있도록 하여 해당 토픽의 필요성이 사라져 삭제하였다.

Topic: `topic/register_hex`

- ESP32에서 publish하여 Node-Red에서 subscribe하는 토픽으로, 리모컨의 적외선 값이 수신되면 payload로 적외선 데이터를 보내는 토픽이다.
- payload의 형식은 IRData를 JSON으로 표시한 것이며, 다음과 같다.

```
{
  "protocol":8,
  "address":18305,
  "command":129,
  "extra":0,
  "numberOfBits":32,,
```

```
  "flags":0,,
  "decodedRawData":2122401665
}
```

이는 esp에서 `handle_ir_rcv()` 에서 수행하며, 해당 함수는 다음과 같이 정의된다.

```
// IR Remote
void handle_ir_rcv() {
  IrReceiver.printIRResultShort(&Serial);
  if (IrReceiver.decodedIRData.rawDataPtr->rawlen < 4) {
    Serial.print("Ignore data with rawlen=");
    Serial.println(IrReceiver.decodedIRData.rawDataPtr->rawlen);
  } else if (IrReceiver.decodedIRData.flags & IRDATA_FLAGS_IS_REPEAT) {
    Serial.println("Ignore repeat");
  } else if (IrReceiver.decodedIRData.flags & IRDATA_FLAGS_IS_AUTO_REPEAT) {
    Serial.println("Ignore autorepeat");
  } else if (IrReceiver.decodedIRData.flags & IRDATA_FLAGS_PARITY_FAILED) {
    Serial.println("Ignore parity error");
  } else if (IrReceiver.decodedIRData.protocol == decode_type_t(0)) {
    Serial.println("Ignore unknown protocol");
  } else {
    // Serialize IRData and publish to MQTT
    JsonDocument payload;

    payload["protocol"] = IrReceiver.decodedIRData.protocol;
    payload["address"] = IrReceiver.decodedIRData.address;
    payload["command"] = IrReceiver.decodedIRData.command;
    payload["extra"] = IrReceiver.decodedIRData.extra;
    payload["numberOfBits"] = IrReceiver.decodedIRData.numberOfBits;
    payload["flags"] = IrReceiver.decodedIRData.flags;
    payload["decodedRawData"] = IrReceiver.decodedIRData.decodedRawData;

    String message = "";
    serializeJson(payload, message);

    int msg_len = message.length() + 1;
    char buf[msg_len];
    message.toCharArray(buf, msg_len);
    client.publish(MQTT_REGISTER_TOPIC, buf);
  }
  IrReceiver.resume();
}
```

IR Data가 수신되면, 올바르게 수신된 IR 신호에 대하여 serialization을 수행하여 string형태로 publish한다.
parity 비트가 올바르지 않거나, 같은 IR 신호가 여러번 입력되거나 올바르지 않은 프로토콜 등의 경우에 대해서 무시하여 처리한다.

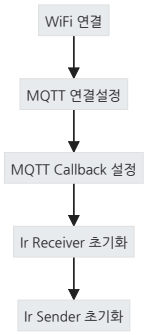
Topic: `topic/control`

- Node-Red에서 publish하여 ESP32에서 subscribe하는 토픽으로, 이전에 저장해둔 리모컨의 적외선 데이터를 보내는 토픽이다.
- payload의 형식은 역시 위와 같이 IRData를 JSON으로 표시한 것이다. 이를 deserialize하기 위해서 esp에서 다음과 같이 deserialization 수행 후, IRData 타입으로 생성한다.

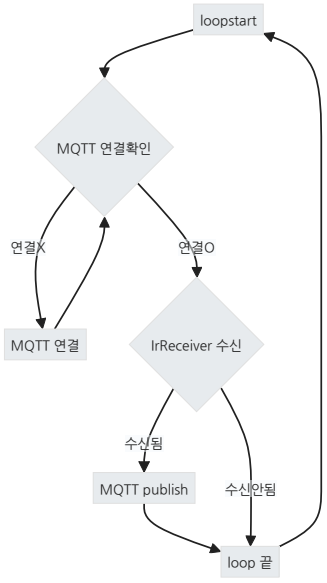
```
// send ir
JsonDocument ir_json;
deserializeJson(ir_json, stMessage);

String tmp_protocol = ir_json["protocol"];
String tmp_address = ir_json["address"];
String tmp_command = ir_json["command"];
String tmp_extra = ir_json["extra"];
String tmp_numberOfBits = ir_json["numberOfBits"];
String tmp_flags = ir_json["flags"];
String tmp_decodedRawData = ir_json["decodedRawData"];

int msg_len = tmp_decodedRawData.length() + 1;
char buf[msg_len];
tmp_decodedRawData.toCharArray(buf, msg_len);

decode_type_t protocol = decode_type_t(tmp_protocol.toInt());
uint16_t address = tmp_address.toInt();
uint16_t command = tmp_command.toInt();
uint16_t extra = tmp_extra.toInt();
uint16_t numberOfBits = tmp_numberOfBits.toInt();
uint8_t flags = tmp_flags.toInt();
IRRawDataType decodedRawData = strtoull(buf, NULL, 10);

IRData ir_data = {
    .protocol = protocol,
    .address = address,
    .command = command,
    .extra = extra,
    .decodedRawData = decodedRawData,
    .numberOfBits = numberOfBits,
    .flags = flags,
};

IrSender.write(&ir_data);
```

Kepler-ESP A 개발과정

Setup



Loop



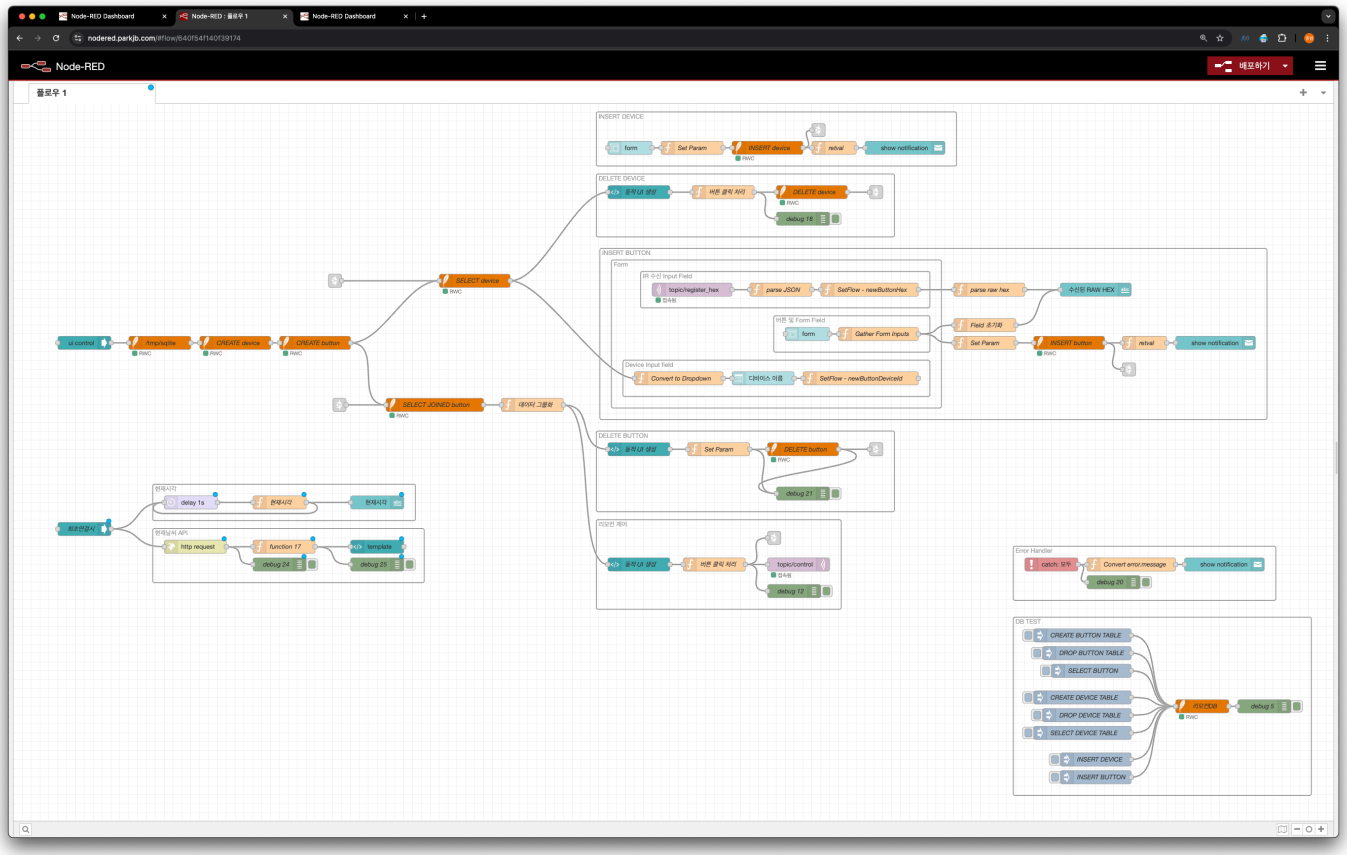
코드 전문

```
#include "buzzer.h"
#include "matrix.h"

#define SEND_PWM_BY_TIMER
#define IR_RECEIVE_PIN 16
#define IR_SEND_PIN 17
#include <IRremote.hpp>

#include <ArduinoJson.h>
#include <PubSubClient.h>
#include <WiFi.h>

WiFiClient espClient;
PubSubClient client(espClient);

#define MQTT_SERVER "broker.emqx.io"
#define MQTT_PORT 1883
#define MQTT_USERNAME "ajouesp"
#define MQTT_PASSWORD "password"

#define MQTT_REGISTER_TOPIC "topic/register_hex"
#define MQTT_CONTROL_TOPIC "topic/control"
#ifdef FEATURE_MATRIX
#define MQTT_MATRIX_TOPIC "topic/matrix"
#define MQTT_FACE_TOPIC "topic/face"
#endif
#ifdef FEATURE_BUZZER
#define MQTT_BUZZER_TOPIC "topic/buzzer"
#endif

char clientId[50];

// WiFi Connection
char SSID_LIST[][20] = {"iPhone", "PARKJB"};
char PASSWORD_LIST[][20] = {"aldkiozs92", "sT?x1s=lp_op"};

void init_wifi() {
  int n = WiFi.scanNetworks();

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < sizeof(SSID_LIST) / sizeof(SSID_LIST[0]); j++) {
      if (WiFi.SSID(i) == SSID_LIST[j]) {
        WiFi.begin(SSID_LIST[j], PASSWORD_LIST[j]);
        Serial.print("[WiFi] Connecting to ");
        Serial.print(SSID_LIST[j]);
        while (WiFi.status() != WL_CONNECTED) {
          Serial.print(".");
          delay(1000);
        }
```

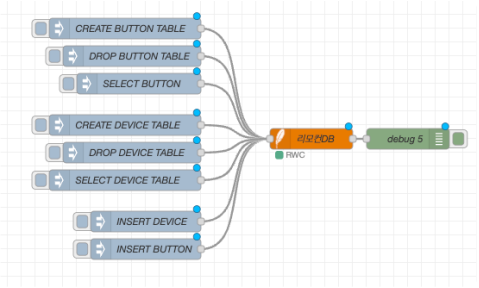
```
        Serial.println();
        Serial.println("[WiFi] Connected to the WiFi network");
        Serial.print("[WiFi] IP Address: ");
        Serial.println(WiFi.localIP());
        return;
      }
    }
  }
}

// MQTT Connection
void connect_mqtt() {
  Serial.print("Attempting MQTT connection...");
  long r = random(1000);
  sprintf(clientId, "clientId-%ld", r);
  if (client.connect(clientId, MQTT_USERNAME, MQTT_PASSWORD)) {
    Serial.print(clientId);
    Serial.println(" connected");
    client.subscribe(MQTT_FACE_TOPIC);
    client.subscribe(MQTT_CONTROL_TOPIC);
#ifdef FEATURE_MATRIX
    client.subscribe(MQTT_MATRIX_TOPIC);
#endif
#ifdef FEATURE_BUZZER
    client.subscribe(MQTT_BUZZER_TOPIC);
#endif
  } else {
    Serial.print("failed, rc=");
    Serial.print(client.state());
    Serial.println(" try again in 5 seconds");
    delay(5000);
  }
}

void mqttCallback(char *topic, byte *message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String stMessage;
  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    stMessage += (char)message[i];
  }
  Serial.println();
#ifdef FEATURE_MATRIX
  if (String(topic) == MQTT_MATRIX_TOPIC) {
    marquee_text(stMessage.c_str());
  }
#endif
#ifdef FEATURE_BUZZER
  if (String(topic) == MQTT_BUZZER_TOPIC) {
    play_canon();
  }
#endif
  if (String(topic) == MQTT_FACE_TOPIC) {
    draw_face(stMessage.toInt());
    Serial.println("Test message received");
  }

  if (String(topic) == MQTT_CONTROL_TOPIC) {
    // send ir
    JsonDocument ir_json;
    deserializeJson(ir_json, stMessage);

    String tmp_protocol = ir_json["protocol"];
    String tmp_address = ir_json["address"];
    String tmp_command = ir_json["command"];
    String tmp_extra = ir_json["extra"];
    String tmp_numberOfBits = ir_json["numberOfBits"];
    String tmp_flags = ir_json["flags"];
    String tmp_decodedRawData = ir_json["decodedRawData"];

    int msg_len = tmp_decodedRawData.length() + 1;
    char buf[msg_len];
    tmp_decodedRawData.toCharArray(buf, msg_len);

    decode_type_t protocol = decode_type_t(tmp_protocol.toInt());
    uint16_t address = tmp_address.toInt();
    uint16_t command = tmp_command.toInt();
    uint16_t extra = tmp_extra.toInt();
    uint16_t numberOfBits = tmp_numberOfBits.toInt();
    uint8_t flags = tmp_flags.toInt();
    IRRawDataType decodedRawData = strtoull(buf, NULL, 10);

    IRData ir_data = {
        .protocol = protocol,
        .address = address,
        .command = command,
        .extra = extra,
        .decodedRawData = decodedRawData,
        .numberOfBits = numberOfBits,
        .flags = flags,
    };

    IrSender.write(&ir_data);
    printIRResultShort(&Serial, &ir_data, false);
  }
}

// IR Remote
void handle_ir_rcv() {
  IrReceiver.printIRResultShort(&Serial);
  if (IrReceiver.decodedIRData.rawDataPtr->rawlen < 4) {
```

```
    Serial.print("Ignore data with rawlen=");
    Serial.println(IrReceiver.decodedIRData.rawDataPtr->rawlen);
  } else if (IrReceiver.decodedIRData.flags & IRDATA_FLAGS_IS_REPEAT) {
    Serial.println("Ignore repeat");
  } else if (IrReceiver.decodedIRData.flags & IRDATA_FLAGS_IS_AUTO_REPEAT) {
    Serial.println("Ignore autorepeat");
  } else if (IrReceiver.decodedIRData.flags & IRDATA_FLAGS_PARITY_FAILED) {
    Serial.println("Ignore parity error");
  } else if (IrReceiver.decodedIRData.protocol == decode_type_t(0)) {
    Serial.println("Ignore unknown protocol");
  } else {
    // Serialize IRData and publish to MQTT
    JsonDocument payload;

    payload["protocol"] = IrReceiver.decodedIRData.protocol;
    payload["address"] = IrReceiver.decodedIRData.address;
    payload["command"] = IrReceiver.decodedIRData.command;
    payload["extra"] = IrReceiver.decodedIRData.extra;
    payload["numberOfBits"] = IrReceiver.decodedIRData.numberOfBits;
    payload["flags"] = IrReceiver.decodedIRData.flags;
    payload["decodedRawData"] = IrReceiver.decodedIRData.decodedRawData;

    String message = "";
    serializeJson(payload, message);

    int msg_len = message.length() + 1;
    char buf[msg_len];
    message.toCharArray(buf, msg_len);
    client.publish(MQTT_REGISTER_TOPIC, buf);
  }
  IrReceiver.resume();
}

void setup() {
  Serial.begin(115200);
  while (!Serial)
    ;
  delay(4000);

#ifdef FEATURE_MATRIX
  init_matrix();
#endif
#ifdef FEATURE_BUZZER
  init_buzzer();
#endif

  // Connect to WiFi
  init_wifi();

  // Connect to MQTT
  client.setServer(MQTT_SERVER, MQTT_PORT);
  client.setCallback(mqttCallback);

  // Initialize IR Receiver
  IrReceiver.begin(IR_RECEIVE_PIN, ENABLE_LED_FEEDBACK);
  Serial.print("Ready to receive IR signals of protocols: ");
  printActiveIRProtocols(&Serial);
  Serial.print("at pin ");
  Serial.println(IR_RECEIVE_PIN);

  // Initialize IR Sender
  IrSender.begin();
  Serial.print("Send IR signals at pin ");
  Serial.println(IR_SEND_PIN);
}

void loop() {
  while (!client.connected())
    connect_mqtt();

  if (IrReceiver.decode())
    handle_ir_rcv();

  client.loop();
}
```

Node-RED 개발과정



디바이스 및 버튼 DB Table 관리 (초기 테스트용)



테이블 생성

```
-- 디바이스 테이블 생성
CREATE TABLE device(
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  device_label TEXT UNIQUE NOT NULL
);

-- 버튼 테이블 생성
CREATE TABLE button(
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  device_id INTEGER NOT NULL,
  button_label TEXT,
  button_hex TEXT NOT NULL,
  FOREIGN KEY(device_id)
  REFERENCES device(id)
  ON DELETE CASCADE
);
```

개별 버튼 요소들은 특정 디바이스에 귀속되기 때문에, button 테이블의 `device_id` 행에서 device 테이블의 `id`와 연결되도록 외래키를 설정하였다.
데이터의 무결성을 위해서 device 테이블 삭제시에 해당 디바이스에 귀속되는 button 까지 삭제할 수 있도록 CASCADE 기법으로 삭제되도록 Foreign key constraint를 설정하였다.

테이블 삭제

```
-- 디바이스 테이블 삭제
DROP TABLE device
-- 버튼 테이블 삭제
DROP TABLE button
```

테이블 확인

```
-- 디바이스 테이블 조회
SELECT * from device
```

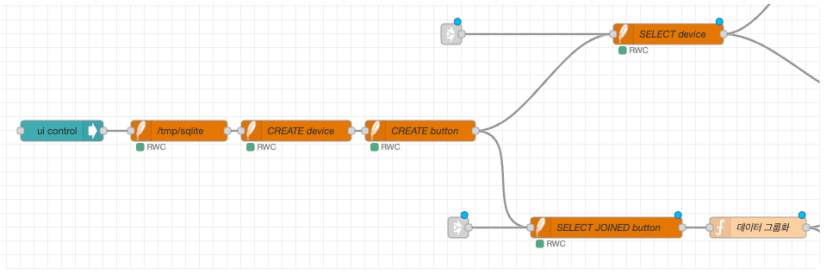
```sql
-- 버튼 테이블 조회
SELECT * from button AS b LEFT OUTER JOIN device AS d ON b.device_id = d.id
```

버튼 테이블을 조회할 때에, 외래키 device_id와 연결된 device의 정보도 표시하기 위해 JOIN을 이용하여 같이 확인할 수 있도록 하였다.
그 결과 다음과 같이 `device_label` 정보도 동시에 조회할 수 있었다.

테이블에 데이터 추가

```sql
-- 디바이스 테이블에 새로운 디바이스 추가
INSERT INTO device(device_label) values("Test Device")
-- 버튼 테이블에 새로운 버튼 추가
INSERT INTO button(device_id, button_label, button_hex) values(1, "Test Button1", "0x11111111")
```

초기데이터 로딩 Flow



노드: CREATE device

```sql
CREATE TABLE IF NOT EXISTS device(
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  device_label TEXT UNIQUE NOT NULL
);
```

노드: CREATE button

```sql
CREATE TABLE IF NOT EXISTS button(
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  device_id INTEGER NOT NULL,
  button_label TEXT,
  button_hex TEXT NOT NULL,
  FOREIGN KEY(device_id)
  REFERENCES device(id)
  ON DELETE CASCADE
);
```

노드: SELECT device

```sql
SELECT * from device;
```

노드: SELECT JOINED button - 데이터 그룹화

노드: SELECT JOINED button

```sql
SELECT
  b.id AS button_id,
  b.button_label,
  b.button_hex,
  b.device_id,
  d.id AS device_id,
  d.device_label
FROM
  button AS b
  LEFT OUTER JOIN
  device AS d
ON b.device_id = d.id;
```

button table의 id column과 device table의 id column이 충돌하여 예기치 않게 에러를 마주하였다.

이를 해결하기 위해서 직접 위와같이 column 이름을 변경하여 쿼리하였다.

노드: 데이터 그룹화

각 디바이스 별로 해당하는 버튼들을 하나의 object로 묶어주는 역할을 하는 노드이다.

```
let data = msg.payload;
let devices = {};
data.forEach(row => {
  let deviceName = row.device_label;
  let button = {
    id: row.button_id,
    name: row.button_label,
    hex: row.button_hex
  };

  if (!devices[deviceName]) {
    devices[deviceName] = {
      deviceInfo: {
        id: row.device_id,
        name: deviceName
      },
      buttons: []
    };
  }

  devices[deviceName].buttons.push(button);
});

msg.devices = devices;

return msg;
```

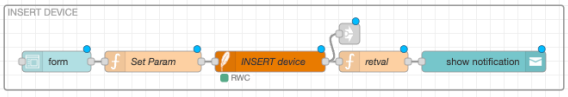
Before (쿼리 직후)


```
payload: array[11]
  [0 … 9]
    0: object
       button_id: 71
       button_label: "전원"
       button_hex: "
       {"protocol":8,"address":18305,"command":129,"extra":0,"numberOf
       Bits":32,"flags":0,"decodedRawData":2122401665}"
       device_id: 15
       device_label: "공청"
    1: object
       button_id: 72
       button_label: "세기"
       button_hex: "
       {"protocol":8,"address":18305,"command":153,"extra":0,"numberOf
       Bits":32,"flags":0,"decodedRawData":1721321345}"
       device_id: 15
       device_label: "공청"
    2: object
       button_id: 73
       button_label: "전원"
       button_hex: "
       {"protocol":20,"address":7,"command":2,"extra":0,"numberOfBits"
       :32,"flags":0,"decodedRawData":4244768519}"
       device_id: 16
       device_label: "티비"
    3: object
       button_id: 74
       button_label: "볼륨업"
       button_hex: "
       {"protocol":20,"address":7,"command":7,"extra":0,"numberOfBits"
       :32,"flags":0,"decodedRawData":4161210119}"
       device_id: 16
       device_label: "티비"
    4: object
```


After 데이터 그룹화


```
msg : Object
object
  payload: array[11]
  tab: 1
  name: "리모컨 설정"
  socketid: "B6F-pYYCSRbt2Lu_AAG3"
  socketip: "210.107.197.179"
  params: object
    empty
  _msgid: "ca2d682429b570fd"
  devices: object
    공청: object
      deviceInfo: object
         id: 15
         name: "공청"
      buttons: array[2]
        0: object
           id: 71
           name: "전원"
           hex: "
           {"protocol":8,"address":18305,"command":129,"extra":0,"number
           OfBits":32,"flags":0,"decodedRawData":2122401665}"
        1: object
           id: 72
           name: "세기"
           hex: "
           {"protocol":8,"address":18305,"command":153,"extra":0,"number
           OfBits":32,"flags":0,"decodedRawData":1721321345}"
    티비: object
      deviceInfo: object
         id: 16
         name: "티비"
      buttons: array[5]
        0: object
        1: object
```


INSERT DEVICE Flow



DB 반영

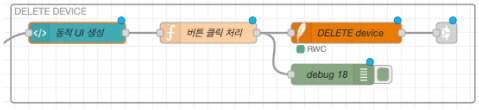
```
const label = msg.payload.device_label
return {
  ...msg,
  "params": {
    $label: label
  }
};
```

```
INSERT INTO device(device_label)
VALUES($label)
RETURNING *;
```

완료 알림 - 노드: retval, 노드: show notification

```
return {
  ...msg,
  payload: `Successfully added Device ${msg.payload[0].device_label}`
};
```

DELETE DEVICE Flow



노드: UI

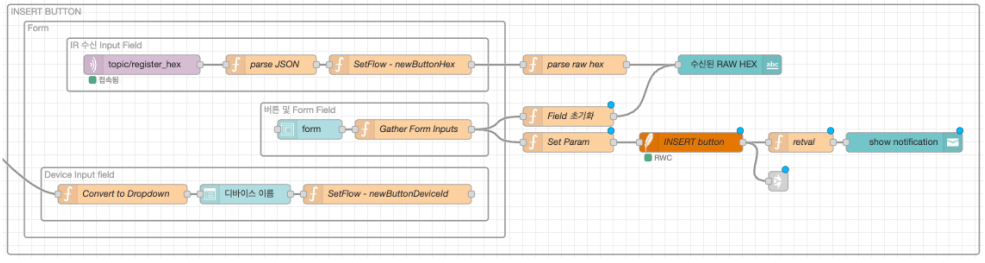
```
<md-card style="min-height: 30vh;">
  <md-card-content>
    <div layout="row" layout-wrap layout-align="center center">
    <div ng-repeat="device in msg.payload" style="width: 96px; margin: 2px;">
      <md-button class="md-raised md-primary" style="width: 100%;" ng-click="send({payload: device})">
      {{device.device_label}}
      </md-button>
    </div>
    </div>
  </md-card-content>
</md-card>

<style>
md-card {
  margin: 10px;
}

md-button {
  margin: 5px;
}
</style>
```

DB 반영

```
return {
  ...msg,
  "params": {
    $device_id: msg.payload.id
  }
};
```

```
DELETE FROM device
WHERE device.id=$device_id;
```

INSERT BUTTON Flow



Form

리모컨으로부터 값 수신 Field

```
return {
  ...msg,
  payload: JSON.parse(msg.payload)
}
```

```
flow.set('newButtonHex', msg.payload)

return msg;
```

디바이스 선택 Dropdown Field

```
let options = [];
msg.payload.forEach(row => {
  let obj = {};
  obj[row.device_label] = row.id;
  options.push(obj);
});

msg.options = options;

return msg;
```

```
flow.set('newButtonDeviceId', msg.payload)
return msg;
```

버튼 이름 Field 및 다른 Field 모아오기

```
let deviceID = flow.get('newButtonDeviceId');
let newButtonHex = flow.get('newButtonHex');
let buttonLabel = msg.payload.button_label

if (deviceID === undefined) {
  throw new Error("버튼을 추가할 디바이스를 선택하세요")
}

if (newButtonHex === undefined) {
  throw new Error("리모컨 HEX를 수신받으세요")
}

msg.payload.device_id = deviceID;
msg.payload.button_hex = newButtonHex
msg.payload.button_label = buttonLabel

return msg;
```

DB 반영

```
const devid = msg.payload.device_id
const hex = msg.payload.button_hex
const label = msg.payload.button_label

return {
  ...msg,
  "params": {
    $device_id: devid,
    $label: label,
    $hex: JSON.stringify(hex)
  }
};
```

```
INSERT INTO button(device_id, button_label, button_hex)
values($device_id, $label, $hex)
RETURNING *;
```

```
return {
  ...msg,
  payload: `Successfully added Button ${msg.payload[0].button_label}`
};
```

Field 초기화

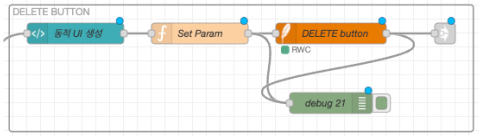
```
flow.set('selectedDeviceValue', undefined)
flow.set('newButtonHex', undefined)

return {
  ...msg,
  payload: ""
};
```

BUTTON UI

```
<md-card style="height: 30vh; padding: 0; margin: 0;">
  <div style="overflow: visible;" ng-repeat="(deviceName, deviceData) in msg.devices">
    <md-card>
```

```
      <md-card-title>
        <md-card-title-text>
          <span class="md-headline">{{deviceName}}</span>
        </md-card-title-text>
      </md-card-title>
      <md-card-content>
        <div layout="row" layout-wrap layout-align="center center">
          <div ng-repeat="button in deviceData.buttons" style="width: 80px; margin: 2px;">
            <md-button class="md-raised md-primary" style="width: 100%;" ng-click="send({payload: button})">{{button.name}}</md-button>
          </div>
        </div>
      </md-card-content>
    </md-card>
  </div>
</md-card>

<style>
md-card {
  margin: 10px;
}

md-button {
  margin: 5px;
}
</style>
```

DELETE BUTTON Flow

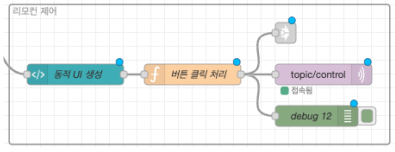


DB 반영

```
return {
  ...msg,
  topic: "del",
  params: {
    $bid: msg.payload.id
  }
};
```

```
DELETE FROM button WHERE id = $bid
```

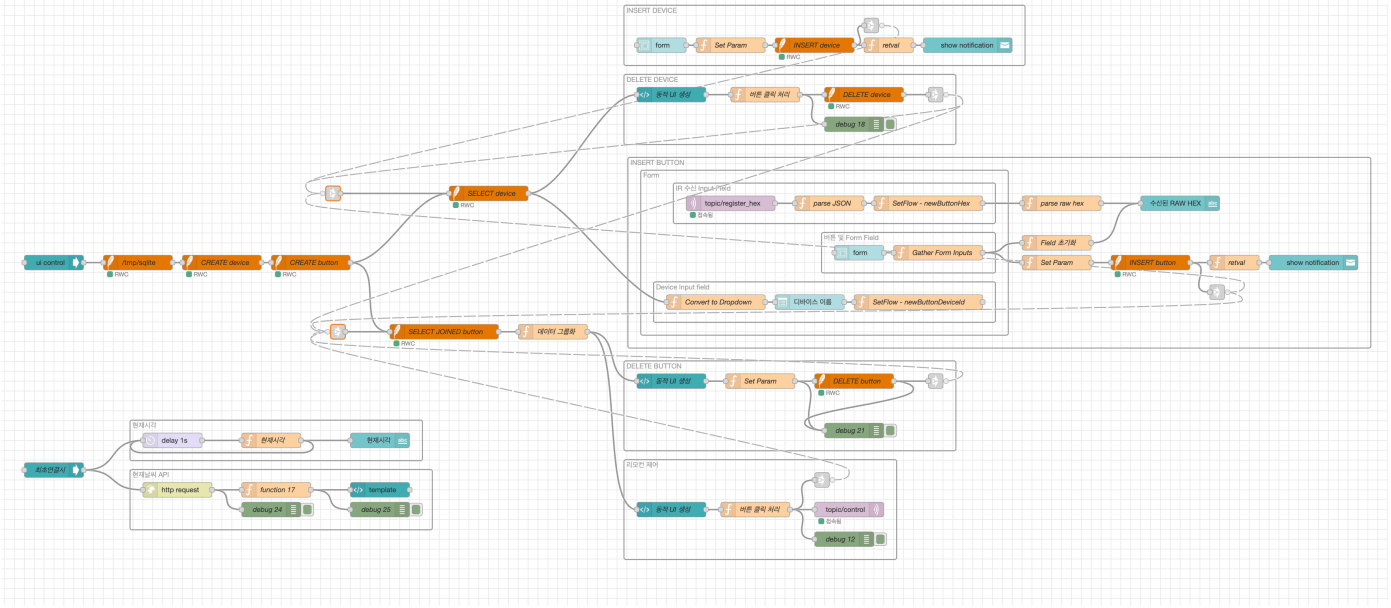
리모컨 제어 Flow




```
return {
  ...msg,
  payload: msg.payload["hex"]
};
```

UI 상태 업데이트

디바이스 혹은 버튼을 추가/삭제 할 시에 이에 맞게 데이터가 다시 로드되어 업데이트 되어야 하기 때문에, 다음과 같이 link를 설정하여 UI를 다시 로드할 수 있도록 업데이트 해주었다.



계획 대비 변동 사항

- 수신모드를 삭제했다. 수신은 항상 받을 수 있도록 설정했으며, UI에 적외선 값이 변경되면 값이 나오도록 설정했다.
- Error발생시, Error의 원인을 디버깅 창과 Notification으로 확인하기 위해 Global Error Handler를 추가했다.
- Matrix를 이용해보고자 관련한 코드를 추가구현해두었으나, 어떻게 응용할지에 대해서 고심해보는 중이다.

개발 중 발생한 문제점 및 해결

- Node-red에 따로 데이터를 저장할 데이터베이스가 없어서 SQlite를 참조했다.
- SQL Statement에 각각 알맞은 코드값들을 넣어주었고, flow값을 넣어주어 버튼이 제 역할을 잘 할 수 있도록 만들어주었다.

개발 결과물 목표 달성 여부

- TV, 무드 등, 스피커, 공기청정기를 IR HUB를 통해 켜고 끌 수 있으며, 다양한 버튼의 HEX 값을 입력 받아 그에 맞는 행동을 디바이스에서 실행되었다.
- 추가적으로 UI에 현재 시각과 날씨를 나타내는 그림을 출력하도록 설정했다.
- 이 IR HUB를 핸드폰으로 접속하면 디바이스들을 제어할 수 있고 추가적으로 현재 시각과 날씨도 알 수 있는 웹사이트를 만들었다고 볼 수 있다.

기대 효과

- 스마트폰을 이용하여 다양한 리모컨으로 제어하는 디바이스를 제어할 수 있다.
- 직관적이고 사용자 친화적인 인터페이스를 통해 기술에 익숙하지 않은 사람도 쉽게 사용할 수 있으며, 외출 중에 집안의 기기를 원격으로 제어할 수 있어 편리성을 높일 것이다.
- IR HUB를 이용하면 삶의 질을 한층 높일 수 있으며, 다른 다양한 장치들과 연동하여 스마트 홈을 구현할 수 있다.
- 실수로 TV나 에어컨을 켜고 나왔을 경우 원격으로 제어가 가능해 에너지 절약효과가 있다.

향후 계획 및 목표

- 온/습도 등의 데이터를 이용하여 실내 공기 질과 온/습도 조절 자동화
- 기상 데이터 API를 이용하여 기상상황에 따른 공기 질과 온/습도 조절 자동화
- 시간에 따른 조명 on/off 기능
- 저렴한 금액으로 스마트 홈 구축가능

서비스 링크 및 계정

- 주소: https://nodered.parkjb.com/ui
- ID: test
- PW: ajouesp008