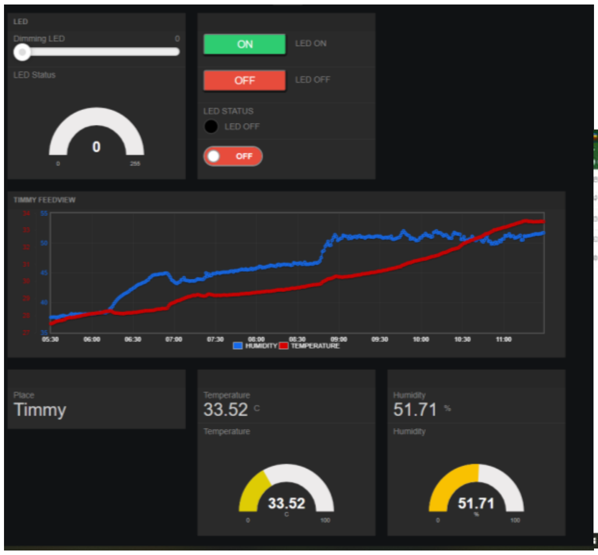
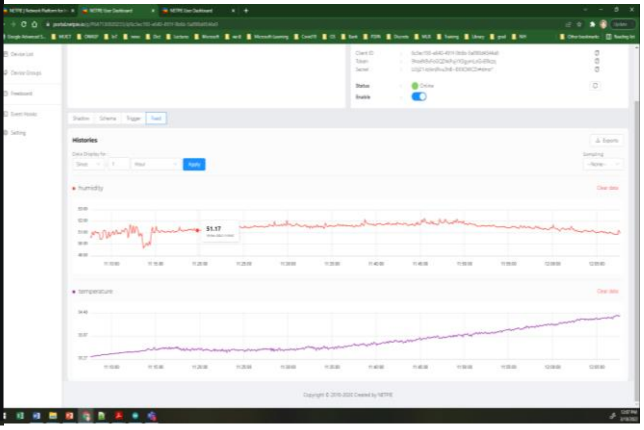
Objectives:

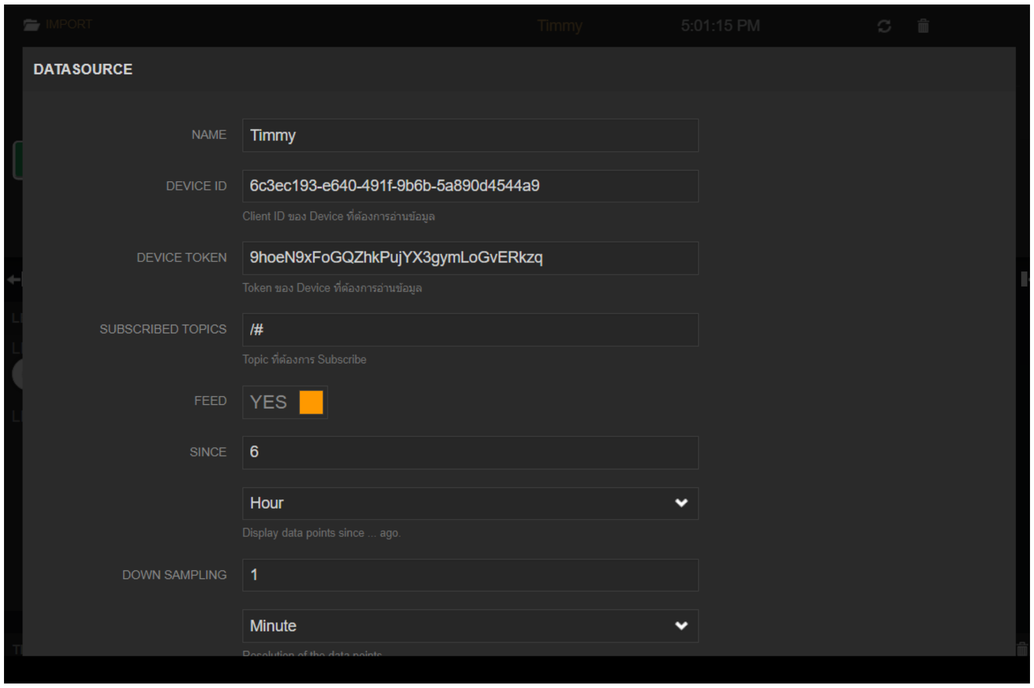
To create NETPIE2020 Dashboard and Freeboard as shown below:





Part 1: Create Data Source (from ESP32-AHT20)

Data from ESP32-AHT20 as a publisher

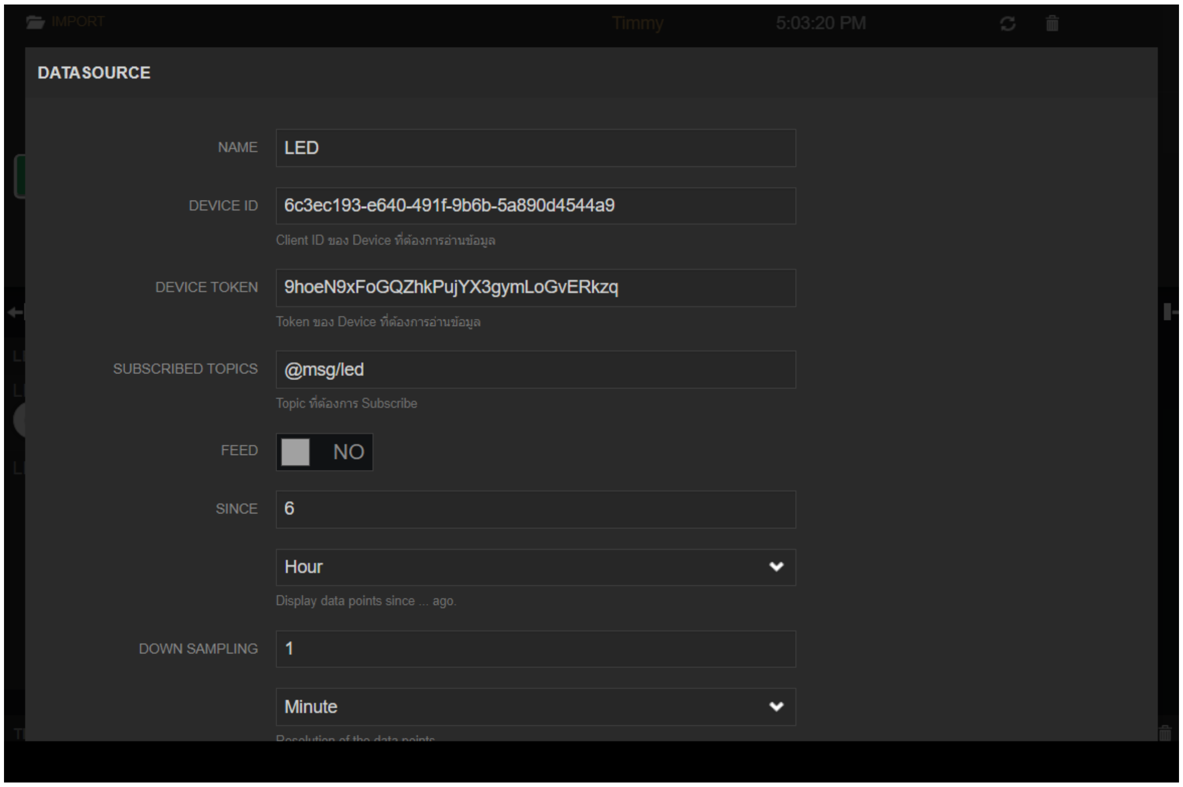


IMPORT Timmy 5:01:15 PM

DATASOURCE

NAME	Timmy
DEVICE ID	6c3ec193-e640-491f-9b6b-5a890d4544a9
	Client ID ของ Device ที่ต้องการอ่านข้อมูล
DEVICE TOKEN	9hoeN9xFoGQZhkPujYX3gymLoGvERkzq
	Token ของ Device ที่ต้องการอ่านข้อมูล
SUBSCRIBED TOPICS	/#
	Topic ที่ต้องการ Subscribe
FEED	YES 🟧
SINCE	6
	Hour
	Display data points since ... ago.
DOWN SAMPLING	1
	Minute
	Resolution of the data points

Part 2: Create Data Source from ESP32-AHT20 as a subscriber



DATASOURCE

NAME	LED
DEVICE ID	6c3ec193-e640-491f-9b6b-5a890d4544a9 Client ID ของ Device ที่ต้องการอ่านข้อมูล
DEVICE TOKEN	9hoeN9xFoGQZhkPujYX3gymLoGvERkzq Token ของ Device ที่ต้องการอ่านข้อมูล
SUBSCRIBED TOPICS	@msg/led Topic ที่ต้องการ Subscribe
FEED	NO
SINCE	6
	Hour Display data points since ... ago.
DOWN SAMPLING	1
	Minute

Schema for ESP32-AHT20 Device

```
{
  "additionalProperties": false,
  "properties": {
    "temperature": {
      "operation": {
        "store": {
          "ttl": "7d"
      },
        "transform": {
          "expression": "($.temperature)"
      }
     },
      "type": "number"
   },
    "humidity": {
      "operation": {
        "store": {
          "ttl": "7d"
      },
        "type": "number"
     }
   }
 }
}
```

Json examples:

Ex 1:

```
{"output": {"temperature":"32.53", "humidity":"52.30"}}
```

Ex 2:

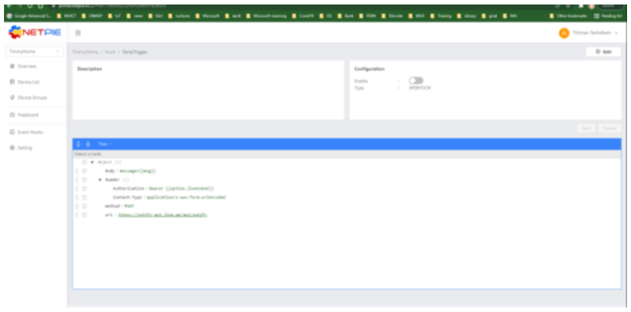
```
{"data":{"Temp":24,"Light":80}}
```

Ex 3:

```
output = "{\"data\": {\"temperature\":" + String(temp.temperature) +
                       ", \"humidity\":" +
String(humidity.relative_humidity) + ",\"place\":\"Timmy\"}}";
{"data": {"temperature":31.37, "humidity":28.03,"place":"Timmy"}}
```

Part 3: Event Hooks

Create "TempTriger" as the same name used for "action" in trigger

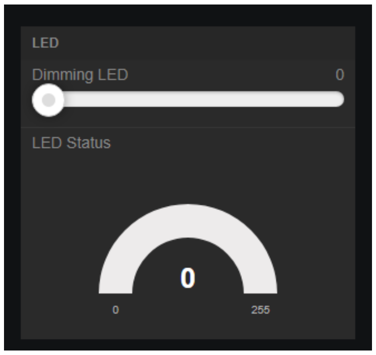


```
{
  "body": "message={{msg}}",
  "header": {
    "Authorization": "Bearer {{option.linetoken}}",
    "Content-Type": "application/x-www-form-urlencoded"
  },
  "method": "POST",
  "uri": "https://notify-api.line.me/api/notify"
}
```

Trigger for "ESP32-AHT20" Device

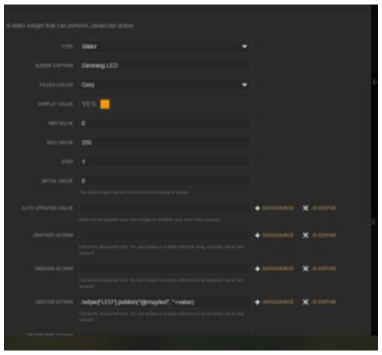
```
{
  "enabled": true,
  "trigger": [
    {
      "action": "TempTrigger",
      "event": "SHADOW.UPDATED",
      "condition": "$.temperature > 27",
      "msg": "My temperature was changed from {{$PREV.temperature}} to
{{$NEW.temperature}}",
      "option": {
        "linetoken": ""
      }
    },
    {
      "action": "TempTrigger",
      "event": "DEVICE.STATUSCHANGED",
      "msg":
"{\"status\":\"{{$NEW.STATUS}}\",\"topic\":\"{{$DEVICEID}}\"}",
      "option": {
        "linetoken": ""
      }
    }
  ]
}
```

Part 4: Dashboard (Widgets)



Widget Slider:

Onstop action: `netpie["LED"].publish("@msg/led", ''+value)`

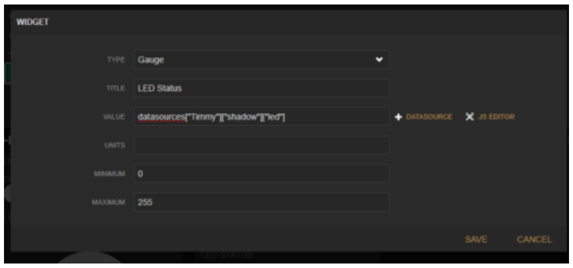

A slider widget that can perform Javascript action.

TYPE	Slider
SLIDER CAPTION	Dimming LED
FILLED COLOR	Grey
DISPLAY VALUE	YES ▪
MIN VALUE	0
MAX VALUE	255
STEP	1
INITIAL VALUE	0
AUTO UPDATED VALUE	
ONSTART ACTION	
ONSLIDE ACTION	
ONSTOP ACTION	netpie["LED"].publish("@msg/led", ''+value)

LED Status:

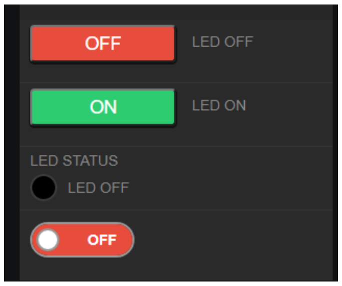
Widget Gauge:

`Value:` `datasources["Timmy"]["shadow"]["led"]`


WIDGET

TYPE	Gauge
TITLE	LED Status
VALUE	datasources["Timmy"]["shadow"]["led"]
UNITS	
MINIMUM	0
MAXIMUM	255

Widget: Button

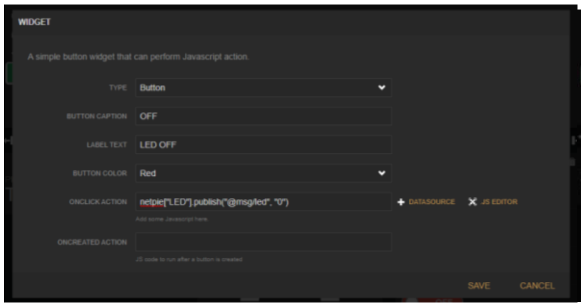


Button OFF:

Onclick Action: `netpie["LED"].publish("@msg/led", "0")`

Button ON:

Onclick Action: `netpie["LED"].publish("@msg/led", "255")`

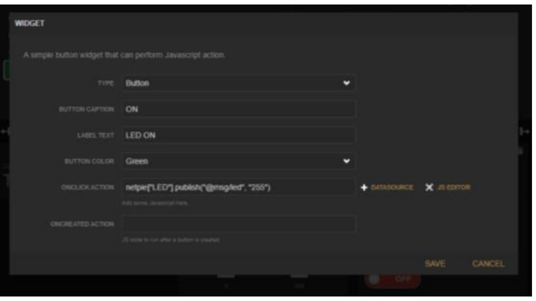

WIDGET

A simple button widget that can perform Javascript action.

TYPE	Button
BUTTON CAPTION	OFF
LABEL TEXT	LED OFF
BUTTON COLOR	Red
ONCLICK ACTION	netpie["LED"].publish("@msg/led", "0")
ONCREATED ACTION	

Add some Javascript here.

JS code to run after a button is created


WIDGET

A simple button widget that can perform Javascript action.

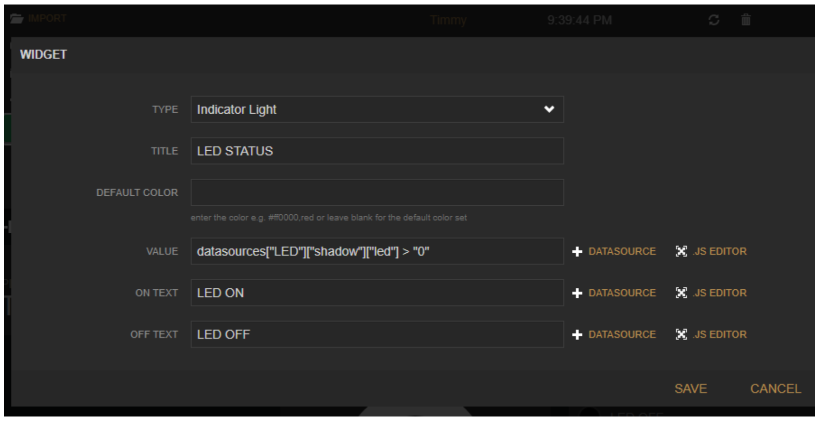
TYPE	Button
BUTTON CAPTION	ON
LABEL TEXT	LED ON
BUTTON COLOR	Green
ONCLICK ACTION	netpie["LED"].publish("@msg/led", "255")
ONCREATED ACTION	

Add some Javascript here.

JS code to run after a button is created

Widget Indicator Light:

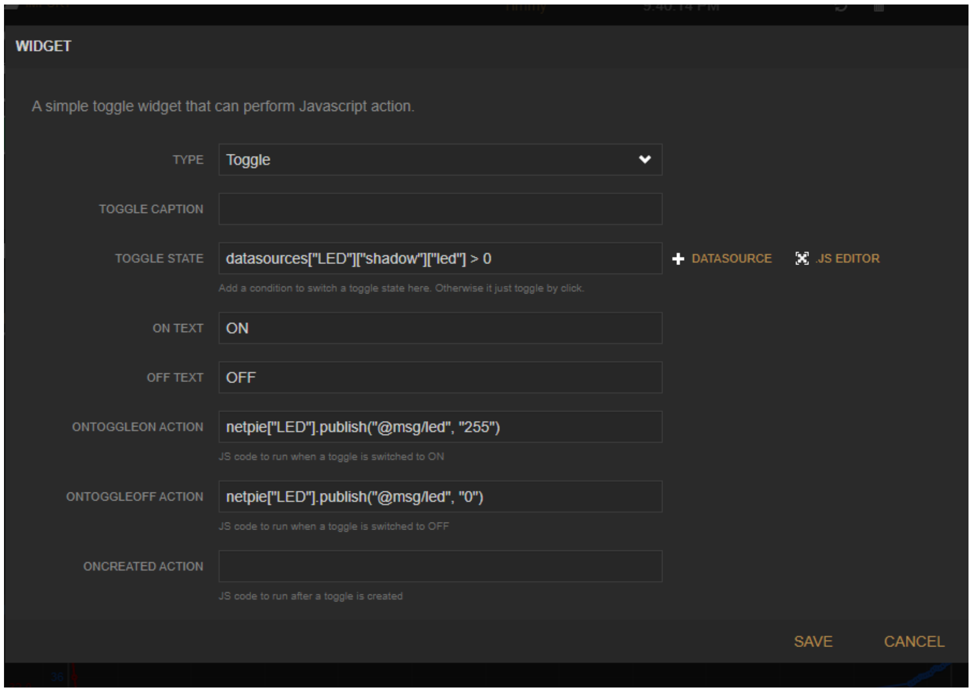
Value: `datasources["LED"]["shadow"]["led"]>"0"`


WIDGET

TYPE	Indicator Light
TITLE	LED STATUS
DEFAULT COLOR	
VALUE	datasources["LED"]["shadow"]["led"] > "0"
ON TEXT	LED ON
OFF TEXT	LED OFF

enter the color e.g. #ff0000,red or leave blank for the default color set

Widget Toggle


WIDGET

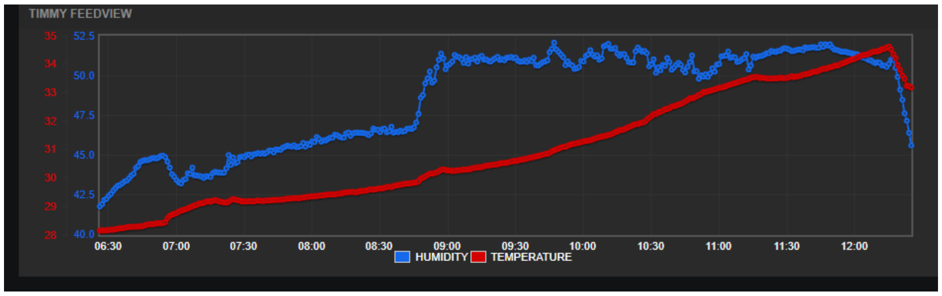
A simple toggle widget that can perform Javascript action.

TYPE	Toggle
TOGGLE CAPTION	
TOGGLE STATE	datasources["LED"]["shadow"]["led"] > 0
ON TEXT	ON
OFF TEXT	OFF
ONTOGGLEON ACTION	netpie["LED"].publish("@msg/led", "255")
ONTOGGLEOFF ACTION	netpie["LED"].publish("@msg/led", "0")
ONCREATED ACTION	

+ DATASOURCE
JS EDITOR
Add a condition to switch a toggle state here. Otherwise it just toggle by click.
JS code to run when a toggle is switched to ON
JS code to run when a toggle is switched to OFF
JS code to run after a toggle is created
SAVE
CANCEL

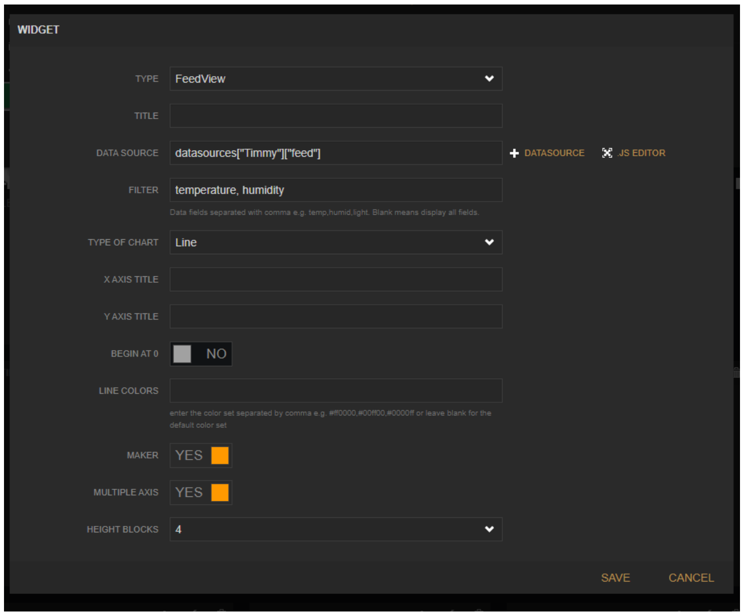
```
TOGGLE STATE: datasources["LED"]["shadow"]["led"] > 0
Ontoggleon action: netpie["LED"].publish("@msg/led", "255")
Ontoggleoff action: netpie["LED"].publish("@msg/led", "0")
```

Widget FeedView



Data source: `datasources["Timmy"]["feed"]`

Filter: `temperature, humidity`



WIDGET

TYPE: FeedView

TITLE:

DATA SOURCE: datasources["Timmy"]["feed"]

FILTER: temperature, humidity

Data fields separated with comma e.g. temp,humid,light. Blank means display all fields.

TYPE OF CHART: Line

X AXIS TITLE:

Y AXIS TITLE:

BEGIN AT 0: NO

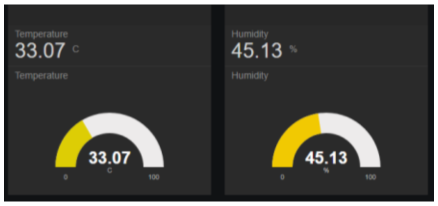
LINE COLORS:

enter the color set separated by comma e.g. #ff0000,#00ff00,#0000ff or leave blank for the default color set

MAKER: YES

MULTIPLE AXIS: YES

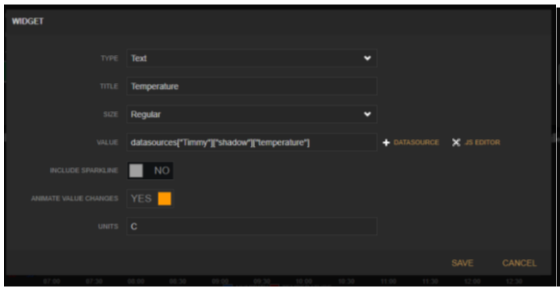
HEIGHT BLOCKS: 4



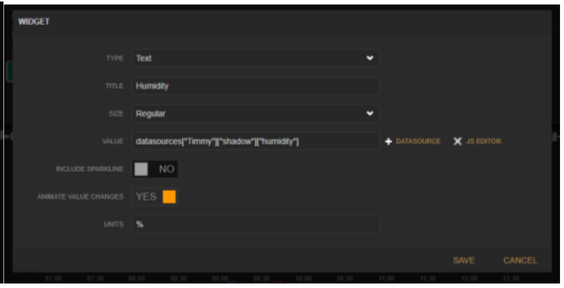
Text Widget:

Value: `datasources["Timmy"]["shadow"]["temperature"]`

Value: `datasources["Timmy"]["shadow"]["humidity"]`

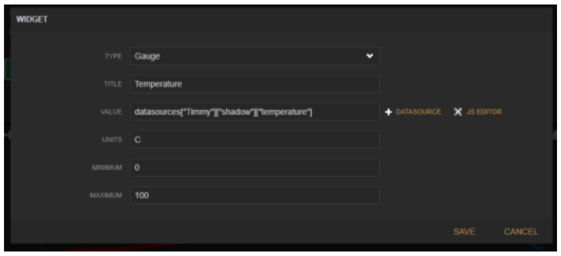

WIDGET

TYPE Text
TITLE Temperature
SIZE Regular
VALUE datasources["Timmy"]["shadow"]["temperature"]
INCLUDE SPARKLINE NO
ANIMATE VALUE CHANGES YES
UNITS C

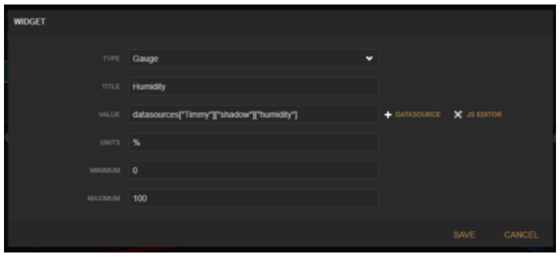

WIDGET

TYPE Text
TITLE Humidity
SIZE Regular
VALUE datasources["Timmy"]["shadow"]["humidity"]
INCLUDE SPARKLINE NO
ANIMATE VALUE CHANGES YES
UNITS %

Gauge Widget:


WIDGET

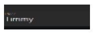
TYPE Gauge
TITLE Temperature
VALUE datasources["Timmy"]["shadow"]["temperature"]
UNITS C
MINIMUM 0
MAXIMUM 100

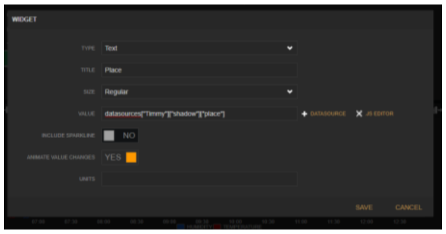

WIDGET

TYPE Gauge
TITLE Humidity
VALUE datasources["Timmy"]["shadow"]["humidity"]
UNITS %
MINIMUM 0
MAXIMUM 100

Text Widget:

Value: `datasources["Timmy"]["shadow"]["place"]`




WIDGET

TYPE Text
TITLE Place
SIZE Regular
VALUE datasources["Timmy"]["shadow"]["place"]
INCLUDE SPARKLINE NO
ANIMATE VALUE CHANGES YES
UNITS

ESP32_AHT20_NETPIE2020.ino

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

#include <Adafruit_AHTX0.h>

#include "config.h"

#define LED 4
#define FREQ 5000
#define LED_CH0 0
#define LED_RES 8

/* MQTT Instance */
WiFiClient espClient;
PubSubClient client(espClient);

bool wifiConnected = true;

Adafruit_AHTX0 aht;
sensors_event_t humidity, temp;



/* Value Buffer */
char buf[200]; //Reserved for 200 bytes
long now, lastMsg;
String output;

void setup() {
  Serial.begin(115200);

  ledcSetup(LED_CH0, FREQ, LED_RES);
  ledcAttachPin(LED, LED_CH0);

  if (! aht.begin()) {
    Serial.println("Could not find AHT? Check wiring");
    while (1) delay(10);
  }
  Serial.println("AHT10 or AHT20 found");

  /* (WiFi) Connection Setup */
  WiFi.mode(WIFI_STA);      // set to station mode
  WiFi.begin(ssid, pass);   // connect to an access point
  delay(5000);

  /* loop until ESP32 can sucesfully connect to the WiFi */
  Serial.printf("Connecting to %s ", ssid);
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }
  /* connection is successful */
  Serial.println(" CONNECTED");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());

  /* MQTT NetPie Server */
  client.setServer(mqttServer, mqttPort); //NetPie server and port
  client.setCallback(callback);
}

void loop() {
  now = millis(); //Milliseconds now (timestamp)
  if(now - lastMsg > 5000){ //Publish new messages to the broker again
when 5s passes; otherwise, let it handle subscribed messages with
little blocking
    lastMsg = now;
    if(!client.connected())
      netpieReconnect();
    client.loop();
    aht.getEvent(&humidity, &temp);// populate temp and humidity
objects with fresh data

    /* NetPie Transmission */
    //sprintf(buf, "{\"data\":{\"temperature\":%.2f,
\"humidity\":%.2f}}", temp.temperature, humidity.relative_humidity);
    //Serial.println(buf);
    output = "{\"data\": {\"temperature\":" + String(temp.temperature)
+ ", \"humidity\":" + String(humidity.relative_humidity) +
",\"place\":\"Timmy\"}}";
    Serial.println(output);
    output.toCharArray(buf, (output.length() + 1));
    client.publish("@shadow/data/update", buf);
    delay(1);
  }
}

void netpieReconnect(){
  while(!client.connected()){
    Serial.println("Connecting to NetPie...");
    if (client.connect(mqttClient, mqttUser, mqttPassword )) {
      Serial.println("connected");
      client.subscribe("@msg/led");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length){
  char msg[length+1];
  memcpy(msg, payload, length);
  msg[length] = '\0';
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.printf(": %s]\n", msg);
  int val = String(msg).toInt();
  int written;

  if(val <= 0)
    written = 0;
  else if(val >= 255)
    written = 255;
  else
    written = val;
  ledcWrite(LED_CH0, written);

  sprintf(buf, "{\"data\":{\"led\":%d}}", written);
  Serial.println(buf);
  client.publish("@shadow/data/update", buf);
//Feedback and record the updated value to NetPie
```

config.h

```
/* NetPie MQTT Server */
const char* mqttServer = "broker.netpie.io";
const int mqttPort = 1883;
const char* mqttClient = ""; //Client ID
const char* mqttUser = ""; //Token
const char* mqttPassword = ""; //Secret

/* (WiFi) Variables */
char ssid[] = ""; // Your WiFi credentials.
char pass[] = ""; // Set password to "" for open networks.
```