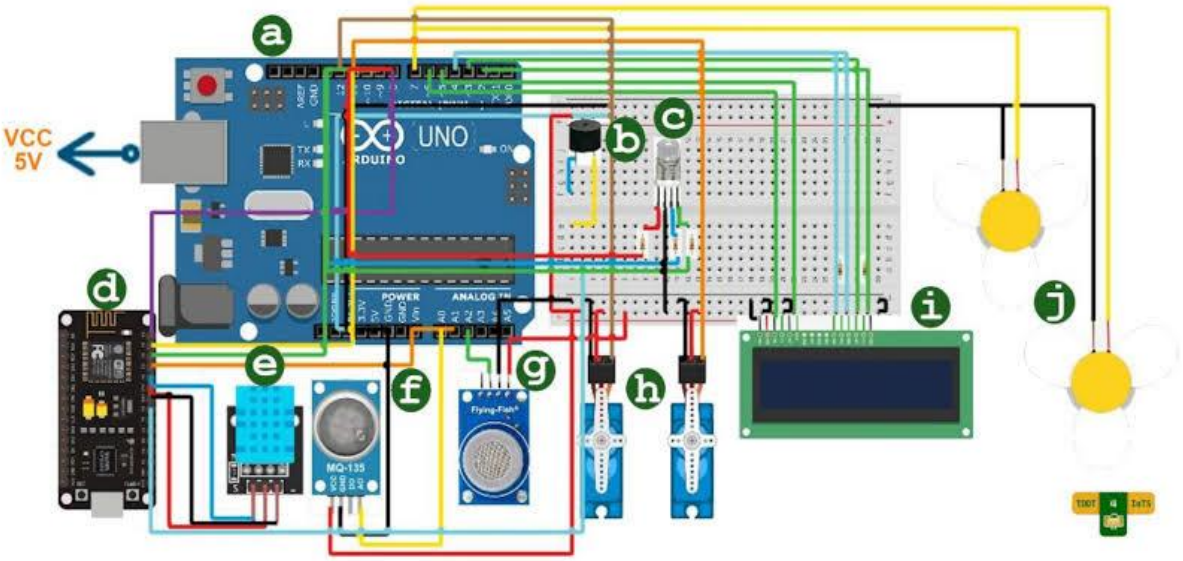
ENVIRONMENTAL MONITORING

PHASE-03 :DEVELOPMENT PART 1

HARDWARE COMPONENTS :

1. Arduino
2. DHT sensors (humidity and temperature)
3. SGP30 (TVOC and eCO2)
4. SDS (Dust sensor)
5. Raspberry Pi
6. Air quality sensors : MQ Series Sensors ,Bosch BME680

HARDWARE OUTLETS DIAGRAM



SOURCE CODE :

```
#include <JSONVar.h>

#include <Arduino_JSON.h>
```

```
#include <JSON.h>
#include <Wire.h>
#include "Adafruit_SGP30.h"
#include "MutichannelGasSensor.h"
#include <ESP8266WiFi.h>
#include <SoftwareSerial.h>
#include "SdsDustSensor.h"
#include "ThingSpeak.h"
#include <Arduino.h>
#include "sensirion_common.h"

#include <Adafruit_Sensor.h>
#include "DHT.h"
// Use this file to store all of the private credentials
// and connection details

#define SECRET_CH_ID1 864649      // replace 0000000 with your channel number
#define SECRET_WRITE_APIKEY1 "D6WO0I37GORJEIV9"   // replace XYZ with your channel write API Key

#define SECRET_CH_ID2 864650      // replace 0000000 with your channel number
#define SECRET_WRITE_APIKEY2 "DDEGP9X1V4WEGEFH"   // replace XYZ with your channel write API Key

#define SECRET_CH_ID3 864651      // replace 0000000 with your channel number
```

```
#define SECRET_WRITE_APIKEY3 "MC5M1BZI4U9422XF"   // replace XYZ with your channel write API Key


#define SECRET_CH_ID4 864652      // replace 0000000 with your channel number

#define SECRET_WRITE_APIKEY4 "1T9T3FK7NR422DJP"   // replace XYZ with your channel write API Key


//#define SECRET_CH_ID1 906528      // replace 0000000 with your channel number

//#define SECRET_WRITE_APIKEY1 "LL4J2EL6WCIW3SKD"   // replace XYZ with your channel write API Key

//

//#define SECRET_CH_ID2 907653      // replace 0000000 with your channel number

//#define SECRET_WRITE_APIKEY2 "JIT20STHHFLPYTBD"   // replace XYZ with your channel write API Key

//

//#define SECRET_CH_ID3 907654      // replace 0000000 with your channel number

//#define SECRET_WRITE_APIKEY3 "XSP8H1C1CD9VDQ2K"   // replace XYZ with your channel write API Key

//

//#define SECRET_CH_ID4 907655      // replace 0000000 with your channel number

//#define SECRET_WRITE_APIKEY4 "Z5HI2QGCMDXMTGQ0"   // replace XYZ with your channel write API Key

int rxPin = 14;

int txPin = 15;

SdsDustSensor sds(rxPin, txPin);
```

```
uint32_t delayMS;


int x;
// ###################### Update the Wifi SSID, Password and IP adress of the server ##########
// WIFI params
char* WIFI_SSID = "JioFi_20FDE31";
char* WIFI_PSWD = "n5v406hr5d";
//char* WIFI_SSID = "WPS unavailable";
//char* WIFI_PSWD = "no game no life";

String CSE_IP      = "onem2m.iiit.ac.in";
// ########################################################

int WIFI_DELAY  = 100; //ms

// oneM2M : CSE params
int   CSE_HTTP_PORT = 80;
String CSE_NAME    = "in-name";
String CSE_M2M_ORIGIN  = "admin:admin";

// oneM2M : resources' params
String DESC_CNT_NAME = "DESCRIPTOR";
String DATA_CNT_NAME = "DATA";
String CMND_CNT_NAME = "COMMAND";
int TY_AE  = 2;
int TY_CNT = 3;
int TY_CI  = 4;
```

```
int TY_SUB = 23;

// HTTP constants
int LOCAL_PORT = 9999;
char* HTTP_CREATED = "HTTP/1.1 201 Created";
char* HTTP_OK    = "HTTP/1.1 200 OK\r\n";
int REQUEST_TIME_OUT = 5000; //ms

//MISC
//int LED_PIN   = D1;/
int SERIAL_SPEED  = 9600;

#define DEBUG

////////////////////////////////////////
//sensor variables
#define DHTPIN 0  // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
DHT dht(DHTPIN, DHTTYPE);
float dht_val[2];
Adafruit_SGP30 sgp;


////////////////////////////////////////
```

```
// Global variables
WiFiServer server(LOCAL_PORT);    // HTTP Server (over WiFi). Binded to listen on LOCAL_PORT contant
WiFiClient client;
String context = "";
String command = "";        // The received command
unsigned long myChannelNumber1 = SECRET_CH_ID1;
const char * myWriteAPIKey1 = SECRET_WRITE_APIKEY1;

unsigned long myChannelNumber2 = SECRET_CH_ID2;
const char * myWriteAPIKey2 = SECRET_WRITE_APIKEY2;
unsigned long myChannelNumber3 = SECRET_CH_ID3;
const char * myWriteAPIKey3 = SECRET_WRITE_APIKEY3;
unsigned long myChannelNumber4 = SECRET_CH_ID4;
const char * myWriteAPIKey4 = SECRET_WRITE_APIKEY4;

String myStatus = "";



// Method for creating an HTTP POST with preconfigured oneM2M headers
// param : url  --> the url path of the targted oneM2M resource on the remote CSE
// param : ty --> content-type being sent over this POST request (2 for ae, 3 for cnt, etc.)
// param : rep  --> the representaton of the resource in JSON format
String doPOST(String url, int ty, String rep) {

  String postRequest = String() + "POST " + url + " HTTP/1.1\r\n" +
```

```
                "Host: " + CSE_IP + ":" + CSE_HTTP_PORT + "\r\n" +
                "X-M2M-Origin: " + CSE_M2M_ORIGIN + "\r\n" +
                "Content-Type: application/json;ty=" + ty + "\r\n" +
                "Content-Length: " + rep.length() + "\r\n"
                "Connection: close\r\n\n" +
                rep;

  // Connect to the CSE address

  Serial.println("connecting to " + CSE_IP + ":" + CSE_HTTP_PORT + "
..");

  // Get a client
  WiFiClient client;
  if (!client.connect(CSE_IP, CSE_HTTP_PORT)) {
    Serial.println("Connection failed !");
    return "error";
  }

  // if connection succeeds, we show the request to be send
#ifdef DEBUG
  Serial.println(postRequest);
#endif

  // Send the HTTP POST request
  client.print(postRequest);

  // Manage a timeout
```

```
  unsigned long startTime = millis();
  while (client.available() == 0) {
    if (millis() - startTime > REQUEST_TIME_OUT) {
      Serial.println("Client Timeout");
      client.stop();
      return "error";
    }
  }

  // If success, Read the HTTP response
  String result = "";
  if (client.available()) {
    result = client.readStringUntil('\r');
    //    Serial.println(result);
  }
  while (client.available()) {
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }
  Serial.println();
  Serial.println("closing connection...");
  return result;
}

// Method for creating an ApplicationEntity(AE) resource on the remote CSE (this is done by sending a POST request)
// param : ae --> the AE name (should be unique under the remote CSE)
String createAE(String ae) {
```

```
  String aeRepresentation =
    "{\"m2m:ae\": {"
    "\"rn\":\"" + ae + "\","
    "\"api\":\"org.demo." + ae + "\","
    "\"rr\":\"true\","
    "\"poa\":[\"http://" + WiFi.localIP().toString() + ":" + LOCAL_PORT + "/" + ae + "\"]"
    "}}";
#ifdef DEBUG
  Serial.println(aeRepresentation);
#endif
  return doPOST("/" + CSE_NAME, TY_AE, aeRepresentation);
}


// Method for creating an Container(CNT) resource on the remote CSE under a specific AE (this is done by sending a POST request)
// param : ae --> the targeted AE name (should be unique under the remote CSE)
// param : cnt  --> the CNT name to be created under this AE (should be unique under this AE)
String createCNT(String ae, String cnt) {
  String cntRepresentation =
    "{\"m2m:cnt\": {"
    "\"rn\":\"" + cnt + "\","
    "\"min\":\"" + -1 + "\""
    "}}";
  return doPOST("/" + CSE_NAME + "/" + ae, TY_CNT, cntRepresentation);
}
```

```
// Method for creating an ContentInstance(CI) resource on the remote CSE under a specific CNT (this is done by sending a POST request)

// param : ae --> the targted AE name (should be unique under the remote CSE)

// param : cnt  --> the targeted CNT name (should be unique under this AE)

// param : ciContent --> the CI content (not the name, we don't give a name for ContentInstances)

String createCI(String ae, String cnt, String ciContent) {

  String ciRepresentation =

    "{\"m2m:cin\": {"

    "\"con\":\"" + ciContent + "\""

    "}}";

  return doPOST("/" + CSE_NAME + "/" + ae + "/" + cnt, TY_CI, ciRepresentation);

}



// Method for creating an Subscription (SUB) resource on the remote CSE (this is done by sending a POST request)

// param : ae --> The AE name under which the SUB will be created .(should be unique under the remote CSE)

//          The SUB resource will be created under the COMMAND container more precisely.

String createSUB(String ae) {

  String subRepresentation =

    "{\"m2m:sub\": {"

    "\"rn\":\"SUB_" + ae + "\","

    "\"nu\":[\"" + CSE_NAME + "/" + ae  + "\"], "

    "\"nct\":1"
```

```
    "}}";
  return doPOST("/" + CSE_NAME + "/" + ae + "/" + CMND_CNT_NAME,
TY_SUB, subRepresentation);
}



// Method to register a module (i.e. sensor or actuator) on a remote oneM2M
CSE
void registerModule(String module, bool isActuator, String
intialDescription, String initialData) {
  if (WiFi.status() == WL_CONNECTED) {
    String result;
    // 1. Create the ApplicationEntity (AE) for this sensor
    result = createAE(module);
    if (result == HTTP_CREATED) {
#ifdef DEBUG
      Serial.println("AE " + module + " created  !");
#endif


      // 2. Create a first container (CNT) to store the description(s) of the
sensor
      result = createCNT(module, DESC_CNT_NAME);
      if (result == HTTP_CREATED) {
#ifdef DEBUG
        Serial.println("CNT " + module + "/" + DESC_CNT_NAME + "
created  !");
#endif
```

```
      // Create a first description under this container in the form of a
ContentInstance (CI)
      result = createCI(module, DESC_CNT_NAME, intialDescription);
      if (result == HTTP_CREATED) {
#ifdef DEBUG
        Serial.println("CI " + module + "/" + DESC_CNT_NAME +
"/{initial_description} created !");
#endif
       }
     }

     // 3. Create a second container (CNT) to store the data  of the sensor
     result = createCNT(module, DATA_CNT_NAME);
     if (result == HTTP_CREATED) {
#ifdef DEBUG
      Serial.println("CNT " + module + "/" + DATA_CNT_NAME + "
created !");
#endif

      // Create a first data value under this container in the form of a
ContentInstance (CI)
      result = createCI(module, DATA_CNT_NAME, initialData);
      if (result == HTTP_CREATED) {
#ifdef DEBUG
        Serial.println("CI " + module + "/" + DATA_CNT_NAME +
"/{initial_aata} created !");
#endif
       }
     }
```

```
    // 3. if the module is an actuator, create a third container (CNT) to store the received commands
    if (isActuator) {
      result = createCNT(module, CMND_CNT_NAME);
      if (result == HTTP_CREATED) {
#ifdef DEBUG
        Serial.println("CNT " + module + "/" + CMND_CNT_NAME + " created !");
#endif

        // subscribe to any ne command put in this container
        result = createSUB(module);
        if (result == HTTP_CREATED) {
#ifdef DEBUG
          Serial.println("SUB " + module + "/" + CMND_CNT_NAME + "/SUB_" + module + " created !");
#endif
        }
      }
    }
  }
}



void init_WiFi() {
  Serial.println("Connecting to  " + String(WIFI_SSID) + " ...");
```

```
  WiFi.persistent(false);
  WiFi.begin(WIFI_SSID, WIFI_PSWD);

  // wait until the device is connected to the wifi network
  while (WiFi.status() != WL_CONNECTED) {
    delay(WIFI_DELAY);
    Serial.print(".");
  }

  // Connected, show the obtained ip address
  Serial.println("WiFi Connected ==> IP Address = " +
WiFi.localIP().toString());
}

void init_HTTPServer() {
  server.begin();
  Serial.println("Local HTTP Server started !");
}

void task_HTTPServer() {
  // Check if a client is connected
  client = server.available();
  if (!client)
    return;

  // Wait until the client sends some data
  Serial.println("New client connected. Receiving request... ");
  while (!client.available()) {
```

```
#ifdef DEBUG_MODE
    Serial.print(".");
#endif
    delay(5);
  }

  // Read the request
  String request = client.readString();
  Serial.println(request);
  client.flush();




  int start, end;
  // identify the right module (sensor or actuator) that received the
notification
  // the URL used is ip:port/ae
  start = request.indexOf("/");
  end = request.indexOf("HTTP") - 1;
  context = request.substring(start + 1, end);
#ifdef DEBUG
  Serial.println(String() + start + " , " + end + " -> " + context + ".");
#endif



  // ingore verification messages
  if (request.indexOf("vrq") > 0) {
    client.flush();
```

```
    return;
  }


  //Parse the request and identify the requested command from the device
  //Request should be like "[operation_name]"
  start = request.indexOf("[");
  end = request.indexOf("]"); // first occurence of
  command = request.substring(start + 1, end);
#ifdef DEBUG
  Serial.println(String() + start + " , " + end + " -> " + command + ".");
#endif

  client.flush();
}

// ######### START OF EXAMPLE ######### //
void init_luminosity() {
  String initialDescription = "Name = LuminositySensor\t"
                              "Unit = Lux\t"
                              "Location = Home\t";
  String initialData = "0";
  registerModule("LuminositySensor", false, initialDescription, initialData);
}

void task_luminosity() {
  int sensorValue;
```

```
  int sensorPin = A0;
  sensorValue = analogRead(sensorPin);
  //sensorValue = random(10, 20);
#ifdef DEBUG
  Serial.println("luminosity value = " + sensorValue);
#endif

  String ciContent = String(sensorValue);

  createCI("LuminositySensor", DATA_CNT_NAME, ciContent);
}



void init_led() {
  String initialDescription = "Name = LedActuator\t"
                              "Location = Home\t";
  String initialData = "off";
  registerModule("LedActuator", true, initialDescription, initialData);
}
//void task_led() {
//
//}
//
//void command_led(String cmd) {
//  //Serial.print(cmd);
//  if (cmd == "switchOn") {
//#ifdef DEBUG
```

```
//    Serial.println("Switching on the LED ...");
//#endif
//    digitalWrite(LED_PIN, LOW);
//  }
//  else if (cmd == "switchOff") {
//#ifdef DEBUG
//    Serial.println("Switching off the LED ...");
//#endif
//    digitalWrite(LED_PIN, HIGH);
//  }
//}
// ######### END OF EXAMPLE ######### //

//////////////////////////////////////////////////////////////////////////////

// ########################################################## //
// ######### USE THIS SPACE TO DECLARE VARIABLES  #########
//
// ########################################################  //

float temp, hum;

// ########################################################  //
void dht_usr()
{  Serial.println("Reading DHT values");
  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
```

```
// Read temperature as Celsius (the default)
float t = dht.readTemperature();
// Read temperature as Fahrenheit (isFahrenheit = true)
float f = dht.readTemperature(true);

// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t) || isnan(f)) {
  ;//Serial.println(F("Failed to read from DHT sensor!"));
  ;
  return;
}

// Compute heat index in Fahrenheit (the default)
float hif = dht.computeHeatIndex(f, h);
// Compute heat index in Celsius (isFahreheit = false)
float hic = dht.computeHeatIndex(t, h, false);
dht_val[0] = h;
dht_val[1] = t;
ThingSpeak.setField(1, String(h));
ThingSpeak.setField(2, String(t));

// write to the ThingSpeak channel
x = ThingSpeak.writeFields(myChannelNumber1, myWriteAPIKey1);
if (x == 200) {
  Serial.println("Channel update successful.");
}
else {
```

```
    Serial.println("Problem updating channel. HTTP error code " + String(x));
  }
  Serial.print(F("Humidity: "));
  Serial.print(h);
  Serial.print(F("%  Temperature: "));
  Serial.print(t);
  Serial.print(F("°C "));
  Serial.print(f);
  Serial.print(F("°F  Heat index: "));
  Serial.print(hic);
  Serial.print(F("°C "));
  Serial.print(hif);
  Serial.println(F("°F"));
}
float sgp_val[10];
void sgp_usr()
{
  if (! sgp.IAQmeasure()) {
    Serial.println("Measurement failed");
    return;
  }
  sgp_val[0] = sgp.TVOC;
  sgp_val[1] = sgp.eCO2;
  Serial.print("TVOC "); Serial.print(sgp.TVOC); Serial.print(" ppb\t");
  Serial.print("eCO2 "); Serial.print(sgp.eCO2); Serial.println(" ppm");
  ThingSpeak.setField(1, String(sgp_val[0]));
  ThingSpeak.setField(2, String(sgp_val[1]));
```

```
  // write to the ThingSpeak channel
  int x = ThingSpeak.writeFields(myChannelNumber2, myWriteAPIKey2);
  if (x == 200) {
    Serial.println("Channel update successful.");
  }
  else {
    Serial.println("Problem updating channel. HTTP error code " + String(x));
  }
  if (! sgp.IAQmeasureRaw()) {
    Serial.println("Raw Measurement failed");
    return;
  }
  //  sgp_val[2]= sgp.rawH2;
  //  Serial.print("Raw H2 "); Serial.print(sgp.rawH2); Serial.print(" \t");
  //  Serial.print("Raw Ethanol "); Serial.print(sgp.rawEthanol);
Serial.println("");


  delay(1000);



  uint16_t TVOC_base, eCO2_base;
  if (! sgp.getIAQBaseline(&eCO2_base, &TVOC_base)) {
    Serial.println("Failed to get baseline readings");
    return;
  }
  Serial.print("****Baseline values: eCO2: 0x"); Serial.print(eCO2_base,
HEX);
```

```
  Serial.print(" & TVOC: 0x"); Serial.println(TVOC_base, HEX);



}

float mgs_val[3];
void mgs_usr()
{
  float c;

  c = gas.measure_NH3();
  mgs_val[0] = c;
  Serial.print("The concentration of NH3 is ");
  if (c >= 0) {
    Serial.print(c);
  }
  else Serial.print("invalid");
  Serial.println(" ppm");

  c = gas.measure_CO();
  mgs_val[1] = c;
  Serial.print("The concentration of CO is ");
  if (c >= 0) {
    Serial.print(c);

  }
  else Serial.print("invalid");
```

```
Serial.println(" ppm");


c = gas.measure_NO2();
mgs_val[2] = c;
Serial.print("The concentration of NO2 is ");
if (c >= 0) {
  Serial.print(c);
}
else Serial.print("invalid");
Serial.println(" ppm");
ThingSpeak.setField(1, String(mgs_val[0]));
ThingSpeak.setField(2, String(mgs_val[1]));
ThingSpeak.setField(3, String(mgs_val[2]));
// write to the ThingSpeak channel
x = ThingSpeak.writeFields(myChannelNumber4, myWriteAPIKey4);
if (x == 200) {
  Serial.println("Channel update successful.");
}
else {
  Serial.println("Problem updating channel. HTTP error code " + String(x));
}
//    c = gas.measure_C3H8();
//    Serial.print("The concentration of C3H8 is ");
//    if(c>=0) {
//      Serial.print(c);
//    }
//    else Serial.print("invalid");
```

```
//    Serial.println(" ppm");
//
//    c = gas.measure_C4H10();
//    Serial.print("The concentration of C4H10 is ");
//    if(c>=0) {
//      Serial.print(c);
//    }
//    else Serial.print("invalid");
//    Serial.println(" ppm");
//
//    c = gas.measure_CH4();
//    Serial.print("The concentration of CH4 is ");
//    if(c>=0) {
//      Serial.print(c);
//    }
//    else Serial.print("invalid");
//    Serial.println(" ppm");
//
//    c = gas.measure_H2();
//    Serial.print("The concentration of H2 is ");
//    if(c>=0) {
//      Serial.print(c);
//    }
//    else Serial.print("invalid");
//    Serial.println(" ppm");
//
//    c = gas.measure_C2H5OH();
```

```
  //    Serial.print("The concentration of C2H5OH is ");
  //    if(c>=0) {
  //      Serial.print(c);
  //    }
  //    else Serial.print("invalid");
  //    Serial.println(" ppm");

  delay(1000);
  ;//Serial.println("...");

}
float sds_val[10];
void sds_usr()
{
  Serial.println("sds");
  PmResult pm = sds.readPm();
  if (pm.isOk()) {
    Serial.print("PM2.5 = ");
    Serial.print(pm.pm25);
    Serial.print(", PM10 = ");
    Serial.println(pm.pm10);
    sds_val[0] = pm.pm25;
    sds_val[1] = pm.pm10;
    ThingSpeak.setField(1, String(pm.pm25));
    ThingSpeak.setField(2, String(pm.pm10));

    // write to the ThingSpeak channel
```

```
    x = ThingSpeak.writeFields(myChannelNumber3, myWriteAPIKey3);
    if (x == 200) {
      Serial.println("Channel update successful.");
    }
    else {
      Serial.println("Problem updating channel. HTTP error code " +
String(x));
    }
  }
}
void setup() {
  // intialize the serial liaison
  Serial.begin(SERIAL_SPEED);

  // Connect to WiFi network
  init_WiFi();
  ThingSpeak.begin(client);
  // Start HTTP server
  init_HTTPServer();
  s16 err;
  u16 scaled_ethanol_signal, scaled_h2_signal;
  // ######### USE THIS SPACE FOR YOUR SETUP CODE ######### //
  Serial.println("sensors initialising!!!");
  sds.begin();
  Serial.println("SDS initialised!!!");
  //  /rial.println(sds.setContinuousWorkingPeriod().toString()); // ensures
sensor has continuous working period - default but not recommended
  dht.begin();
```

```
  Serial.println("dht initialised!!!");
  sgp.begin();
  Serial.println("sgp initialised!!!");
  gas.begin(0x04);//the default I2C address of the slave is 0x04
  Serial.println("gas initialised!!!");
  gas.powerOn();
  Serial.println("gas initialised!!!");



  // ############################################################ //

}

// Main loop of the µController
void loop() {
  //
################################################################
## //
  // ######### USE THIS SPACE FOR YOUR SENSOR POLING CODE
######### //
  //
################################################################
##  //
  Serial.print("yo");
  for(int i=0;i<30;i++)
  { dht_usr();
    sgp_usr();
    mgs_usr();
    sds_usr();
```

```
  delay(20000);
 }
 // ############################################################ //
```

```
 //
############################################################
###### //
 // ######### USE THIS SPACE FOR YOUR SENDING DATA TO
SERVER  ######### //
 //
############################################################
###### //
```

```
 /*
    CreateCI(AE_NAME,CONTAINER_NAME,SENSOR_VALUE)

    CreateCI is what posts your sensor data into the container.

    In the below example:
        AE_NAME: Team8_Automated_Driving (As stated in the resource
tree)
        CONTAINER_NAME : node_1 ( or as stated in the resource tree)
        SENSOR_VALUE : string of comma separated all sensor values (Eg:
27,25 is temp,hum)
 */
```

```
  // Storing as a string in a single containers

  String sensor_value_string;

  sensor_value_string = String("dht : ") + String(dht_val[0]) + String(" , ") +
String(dht_val[1]) + String(" \n ") + String("sgp : ") + String(sgp_val[0]) +
String(" , ") + String(sgp_val[1]) + String(" \n ") + String("multi : ") +
String(mgs_val[0]) + String(" , ") + String(mgs_val[1]) + String(" , ") +
String(mgs_val[2]) + String(" \n ") + String("sds : ") + String(sds_val[0]) +
String(" , ") + String(sds_val[1]) + String(" \n ") ;

  createCI("Team14_Indoor_air_pollution_Mess", "node_1",
sensor_value_string);


  // Check if the data instance was created.

//  delay(600000); // DO NOT CHANGE THIS VALUE



  //
################################################################
###### //

}
```