

SPRINT -1

GAS LEAKAGE MONITORING AND ALERTING SYSTEM

Team ID	PNT2022TMID39130
Project Name	Gas Leakage Monitoring and Alerting System for Industries

SIMULATION CREATION USING WOKWI:

CODE:

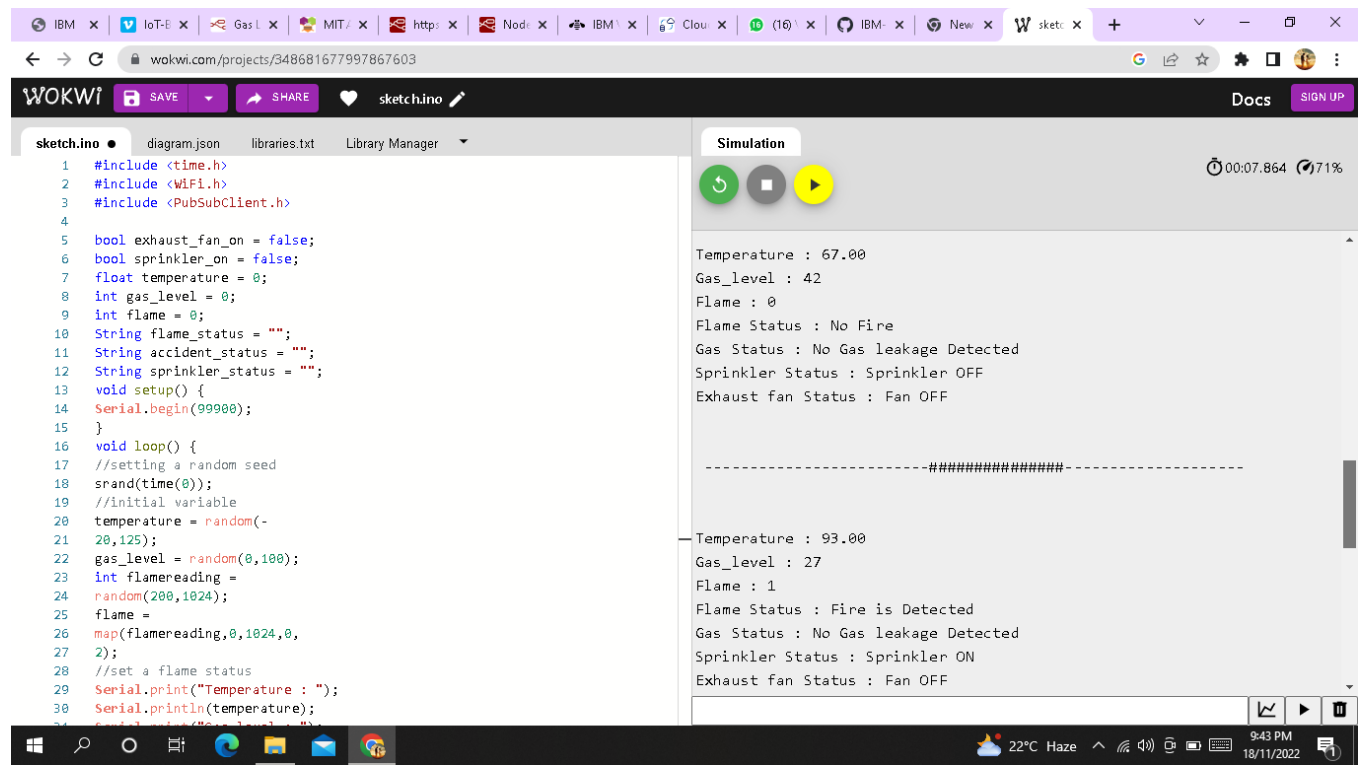
```
#include <time.h>
#include <WiFi.h>
#include <PubSubClient.h>
bool exhaust_fan_on = false;
bool sprinkler_on = false;
float temperature = 0;
int gas_level = 0;
int flame = 0;
String flame_status = "";
String accident_status = "";
String sprinkler_status = "";
void setup() {
Serial.begin(99900);
}
void loop() {
//setting a random seed
srand(time(0));
//initial variable
temperature = random(-
20,125);
gas_level = random(0,1000);
int flamereading =
random(200,1024);
flame =
map(flamereading,0,1024,0,
2);
//set a flame status
Serial.print("Temperature : ");
Serial.println(temperature);
Serial.print("Gas_level : ");
Serial.println(gas_level);
Serial.print("Flame : ");
Serial.println(flame);
switch (flame) {
case 0:
```

```

flame_status = "No Fire";
Serial.println("Flame Status : "+flame_status);
break;
case 1:
flame_status = "Fire is Detected";
Serial.println("Flame Status : "+flame_status);
break;
}
//Gas Detection
if(gas_level > 100){
Serial.println("Gas Status : Gas leakage Detected");
}
else{
exhaust_fan_on = false;
Serial.println("Gas Status : No Gas leakage Detected");
}
//send the sprinkler status
if(flame){
sprinkler_status =
"Sprinkler ON";
Serial.println("Sprinkler Status : "+sprinkler_status);
}
else{
sprinkler_status = "Sprinkler OFF";
Serial.println("Sprinkler Status : "+sprinkler_status);
}
//toggle the fan according to gas
if(gas_level > 100){
exhaust_fan_on = true;
Serial.println("Exhaust fan Status : Fan ON");
}
else{
exhaust_fan_on = false;
Serial.println("Exhaust fan Status : Fan OFF");
}
Serial.println("");
Serial.println("");
Serial.println("-----#####-----");
Serial.println("");
Serial.println("");
delay(1000);
}

```

SIMULATION OUTPUT:



The screenshot shows the Wokwi web-based IDE. On the left, the `sketch.ino` file is open, displaying C++ code for an IoT simulation. The code includes libraries for `time`, `WiFi`, and `PubSubClient`. It defines variables for `exhaust_fan_on`, `sprinkler_on`, `temperature`, `gas_level`, `flame`, `flame_status`, `accident_status`, `sprinkler_status`, and `exhaust_fan_status`. The `setup` function initializes `Serial` at 999000. The `loop` function sets a random seed, initializes `temperature` to a random value between 20 and 125, `gas_level` to a random value between 0 and 100, and `flame` to a random value between 200 and 1024. It then maps the `flame` value to a `flamereading` between 0 and 1024, and sets the `flame` status based on the `flamereading`. The `Serial` port prints the current `temperature` and `flame` status.

On the right, the **Simulation** panel shows the current state of the simulation. It includes a play button and a stop button. The simulation output shows the following data:

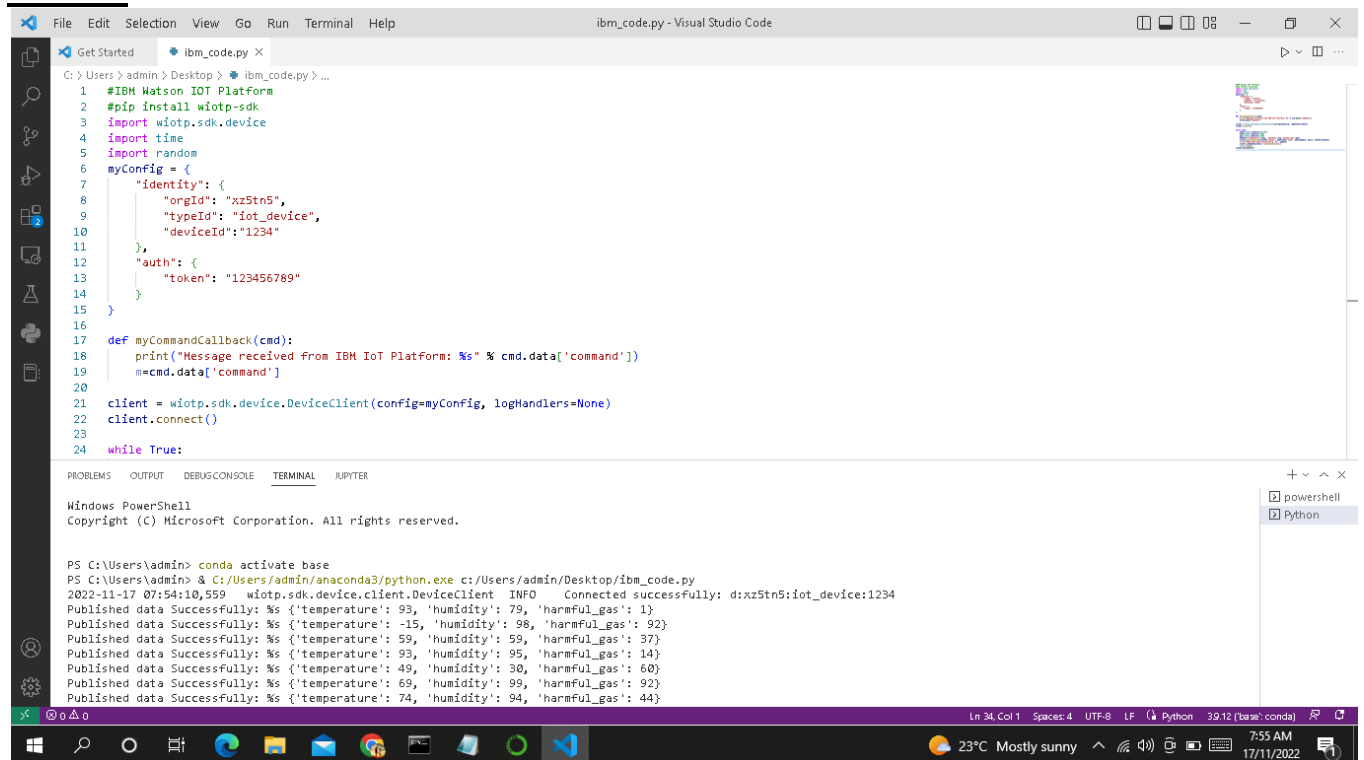
```
Temperature : 67.00
Gas_level : 42
Flame : 0
Flame Status : No Fire
Gas Status : No Gas leakage Detected
Sprinkler Status : Sprinkler OFF
Exhaust fan Status : Fan OFF
```

Below this, a separator line is shown, followed by the next state of the simulation:

```
Temperature : 93.00
Gas_level : 27
Flame : 1
Flame Status : Fire is Detected
Gas Status : No Gas leakage Detected
Sprinkler Status : Sprinkler ON
Exhaust fan Status : Fan OFF
```

CONNECTING IBM CLOUD USING PYTHON CODE:

CODE:



The screenshot shows the Visual Studio Code editor with a Python file named `ibm_code.py`. The code is designed to connect to the IBM Watson IoT Platform and publish data. It includes the following components:

- Imports:** `wiotp.sdk.device` and `time` are imported.
- Configuration:** A `myConfig` dictionary is defined with the following values:
 - `identity`: `{ "orgId": "xz5tn5", "typeId": "iot_device", "deviceId": "1234" }`
 - `auth`: `{ "token": "123456789" }`
- Callback Function:** A `myCommandCallback` function is defined, which prints the received command and updates the `cmd.data['command']` field.
- Client Initialization:** A `DeviceClient` is created using the `myConfig` dictionary and the `myCommandCallback` function. The client is then connected to the IoT Platform.
- Loop:** A `while True:` loop is used to continuously publish data to the IoT Platform.

The terminal output shows the following messages:

```
PS C:\Users\admin> conda activate base
PS C:\Users\admin> & C:/Users/admin/anaconda3/python.exe c:/Users/admin/Desktop/ibm_code.py
2022-11-17 07:54:10,559 |wiotp.sdk.device.client.DeviceClient| INFO | Connected successfully: d:xz5tn5:iot_device:1234
Published data Successfully: %s {'temperature': 93, 'humidity': 79, 'harmful_gas': 1}
Published data Successfully: %s {'temperature': -15, 'humidity': 98, 'harmful_gas': 92}
Published data Successfully: %s {'temperature': 59, 'humidity': 59, 'harmful_gas': 37}
Published data Successfully: %s {'temperature': 93, 'humidity': 95, 'harmful_gas': 14}
Published data Successfully: %s {'temperature': 49, 'humidity': 30, 'harmful_gas': 60}
Published data Successfully: %s {'temperature': 69, 'humidity': 99, 'harmful_gas': 92}
Published data Successfully: %s {'temperature': 74, 'humidity': 94, 'harmful_gas': 44}
```

OUTPUT IN IBM CLOUD:

