BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP. HCM

INDUSTRIAL UNIVERSITY OF HOCHIMINH CITY

CHUYÊN ĐỀ IOT

BÀI BÁO CÁO 11 - 12

NHÓM 3

Giảng viên : **CAO VĂN KIÊN**

Sinh viên :

Trần Công Hòa	20017691
Lê Tấn Tài	20027041
Phạm Gia Bách	20026331

TP.HCM – 2023

Báo cáo thí nghiệm buổi 11-12: Xây dựng ứng dụng IOT
Base: 3Đ

- 1 Raspberry Pi
- Web Node-red cơ bản
- Server Thingspeak
- Chương trình tự chạy khi cắm nguồn
- Điều khiển 2 LED; Giám sát 2 giá trị nhiệt độ, độ ẩm

Things:

- 2 x Cơ cấu chấp hành không phải on/off: **+1 (Things +2 ↑)**
- 2 x Cơ cấu chấp hành/ cảm biến không phải on/off: **+1**
- Raspberry TCP/UDP có 1 cảm biến, 1 cơ cấu chấp hành tự chọn: **+ 1**

Server:

- Server tự viết có CSDL: **+1 (Things +1 ↑)**
- Server hỗ trợ cả HTTP API và MQTT trong đọc/ghi dữ liệu, dữ liệu gửi lên qua HTTP API phải kích hoạt cơ chế Subscribe trong MQTT. **+1 (Things +3 ↑)**

Web:

- Có thêm Tab thể hiện dữ liệu dạng đồ thị lấy từ CSDL thông qua HTTP API: **+1 (Things +1 ↑)**
- Giao diện quản lý thiết bị hiển thị các thông tin trạng thái kết nối, giá trị dữ liệu cuối cùng gửi lên server, thời gian dữ liệu cuối cùng gửi lên: **+1 (Things +2 ↑)**

I. Code chương trình

❖ ***fastAPI***

```
import uvicorn
import paho.mqtt.client as mqtt
from fastapi import FastAPI
from pydantic import BaseModel
from typing import Optional
import pymongo
import json

app = FastAPI()

myclient = 
pymongo.MongoClient("mongodb+srv://Loo:24@loo.isuovt2.mongodb.net/?retryWrites=tr
ue&w=majority")
mydb = myclient["DataBase"]
mycol = mydb["LooData"]

def on_connect(client, userdata, flags, rc):
    print("Connected with Result Code {}".format(rc))

def on_disconnect(client, userdata, rc):
    print("Disconnected from Broker")
```

```
client = mqtt.Client("Loo_fast")
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.username_pw_set(username="Loo3", password="242002")
client.connect("192.168.1.26", 1883, 60)

class Item(BaseModel):
    id: Optional[int]
    time: Optional[str]
    temp: Optional[float]
    humi: Optional[float]
    relay: Optional[float]
    space: Optional[float]
    moi: Optional[float]
    light: Optional[float]
    lcd: Optional[str]
    ledstick: Optional[float]
    led1: Optional[float]
    led2: Optional[float]

def mqttJson(data:dict):
    dictData = {
        "id": data["id"],
        "time": data["time"],
        "temp": data["temp"],
        "humi": data["humi"],
        "relay": data["relay"],
        "space": data["space"],
        "moi": data["moi"],
        "light": data["light"],
        "lcd": data["lcd"],
        "ledstick": data["ledstick"],
        "led1": data["led1"],
        "led2": data["led2"]
    }
    jsonData = json.dumps(dictData)
    client.publish("Loo/publish", jsonData)

@app.post("/update_post")
async def update_data_post(item: Item):
    myDict = {
        "id": item.id,
        "time": item.time,
        "temp": item.temp,
```

```python
        "humi": item.humi,
        "relay": item.relay,
        "space": item.space,
        "moi": item.moi,
        "light": item.light,
        "lcd": item.lcd,
        "ledstick": item.ledstick,
        "led1": item.led1,
        "led2": item.led2
    }
    print("update_post: {}".format(myDict))
    mycol.insert_one(myDict)
    return {"ok"}

@app.get("/getdata")
async def get_data():
    x = mycol.find().sort("_id", -1).limit(1)[0]
    print("get: {}".format(x))
    data_return  = {
        "id": x["id"],
        "time": x["time"],
        "temp": x["temp"],
        "humi": x["humi"],
        "relay": x["relay"],
        "space": x["space"],
        "moi": x["moi"],
        "light": x["light"],
        "lcd": x["lcd"],
        "ledstick": x["ledstick"],
        "led1": x["led1"],
        "led2": x["led2"]
    }
    return data_return

@app.post("/updateUrl")
async def update_data(item: Item):
    dictData = {
        "id": item.id,
        "time": item.time,
        "temp": item.temp,
        "humi": item.humi,
        "relay": item.relay,
        "moi": item.moi,
        "light": item.light,
        "space": item.space,
        "lcd": item.lcd,
        "ledstick": item.ledstick,
        "led1": item.led1,
        "led2": item.led2
    }
    mycol.insert_one(dictData)
    mqttJson(dictData)
    return {"ok"}

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

❖ ***udpServer***

```
import socket
import ast
import paho.mqtt.client as mqtt
import json
import datetime
import Adafruit_DHT
from gpiozero import LED

#Init sensors
sensor = Adafruit_DHT.DHT11
gpio = 5
relay = LED(16)

# Init broker
localIP = "0.0.0.0"
localPort = 8000
bufferSize = 1024

# Init UDP
udpServer = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
udpServer.bind((localIP, localPort))
print("UDP server up and listening")

# Init mqtt
client = mqtt.Client("LooUp")
client.username_pw_set(username="Loo3", password="242002")
client.connect("192.168.1.86", 1883, 60)

# Init color text
txtReset = "\033[0m"
```

```
txtGreen = "\033[32m"

# Init data
currentTimeFirst = datetime.datetime.now()
formattedTimeFirst = currentTimeFirst.strftime('%Y-%m-%d %H:%M:%S')

#Init sensor
dataRasp2 = {
    "id": 0,
    "moi": 0,
    "light": 0,
    "space": 0,
    "lcd": 0,
    "ledstick": 0,
    "led1": 0,
    "led2": 0
}

dataRasp1 = {
    "time": formattedTimeFirst,
    "humi": 0,
    "temp": 0,
    "relay": 0
}

dataUpdate={
    "lcd": 0,
    "ledstick": 0,
    "led1": 0,
    "led2": 0
}

# Init function
def mergedData(data1, data2):
    merged_dict = {**data1, **data2}
    return merged_dict

def processReceived(dataReceived):
    global dataRasp2
    global dataRasp1
    input_string = dataReceived.decode("utf-8")
    dataDict = ast.literal_eval(input_string)
    dataRasp2["id"] = dataDict["id"]
    dataRasp2["moi"] = dataDict["moi"]
    dataRasp2["light"] = dataDict["light"]
```

```python
    dataRasp2["space"] = dataDict["space"]
    mergedRasp = mergedData(dataRasp1, dataRasp2)
    return mergedRasp

def sendJsonData(data: dict):
    jsonData = json.dumps(data)
    client.publish("Loo/publish", jsonData)

def on_connect(client, userdata, flags, rc):
    print("Connected With Result Code {}".format(rc))

def on_disconnect(client, userdata, rc):
    print("Disconnected From Broker")

def on_message(client, userdata, message):
    global dataUpdate
    global dataRasp2
    if message.topic == "Loo/subscribed/lcd":
        if message.payload.decode() != 'null':
            dataUpdate["lcd"] = message.payload.decode()
            dataRasp2["lcd"] = message.payload.decode()
            print("LCD: {}".format(dataUpdate["lcd"]))
    if message.topic == "Loo/subscribed/ledstick":
        if message.payload.decode() != 'null':
            dataUpdate["ledstick"] = int(message.payload.decode())
            dataRasp2["ledstick"] = int(message.payload.decode())
            print("Led Stick: {}".format(dataUpdate["ledstick"]))
    if message.topic == "Loo/subscribed/led1":
        if message.payload.decode() != 'null':
            dataUpdate["led1"] = int(message.payload.decode())
            dataRasp2["led1"] = int(message.payload.decode())
            print("Led1: {}".format(dataUpdate["led1"]))
    if message.topic == "Loo/subscribed/led2":
        if message.payload.decode() != 'null':
            dataUpdate["led2"] = int(message.payload.decode())
            dataRasp2["led2"] = int(message.payload.decode())
            print("Led 2: {}".format(dataUpdate["led2"]))
    if message.topic == "Loo/subscribed/relay":
        if message.payload.decode() != 'null':
            dataRasp1["relay"] = int(message.payload.decode())
            print("relay: {}".format(dataRasp1["relay"]))

def relayFun():
    global dataRasp1
    if dataRasp1["relay"] == 1:
        relay.on()
    elif dataRasp1["relay"] == 0:
        relay.off()


client.subscribe("Loo/subscribed/relay")
client.subscribe("Loo/subscribed/lcd")
client.subscribe("Loo/subscribed/ledstick")
client.subscribe("Loo/subscribed/led1")
client.subscribe("Loo/subscribed/led2")
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.on_message = on_message
client.loop_start()

while True:
    try:
        udpServer.settimeout(1)
        humi, temp = Adafruit_DHT.read_retry(sensor, gpio)
        dataRasp1["temp"] = temp
        dataRasp1["humi"] = humi
        relayFun()

        print(txtGreen + "Tác vụ nhận từ Slave" + txtReset)
        currentTime = datetime.datetime.now()
        formattedTime = currentTime.strftime('%Y-%m-%d %H:%M:%S')
        dataRasp2["time"] = formattedTime

        msgToClient = str("ok")
        bytesToSend = str.encode(msgToClient)
        ClientMsg = udpServer.recvfrom(bufferSize)
        ClientMsg_message = ClientMsg[0]
        clientMsgAddress = ClientMsg[1]

        dataRev = processReceived(ClientMsg_message)

        message = "Mesage from Client: {}".format(dataRev)
        address = "Client IP + Port: {}".format(clientMsgAddress)
        print(address)
        print(message)

        sendJsonData(dataRev)

        udpServer.sendto(bytesToSend, clientMsgAddress)
        print(dataUpdate)
        print("----------------------------------------------------------------")

        print(txtGreen + "Tác vụ gửi đến slave"+txtReset)
        msgToServer = str(dataUpdate)
        bytesTServer = str.encode(msgToServer)

        udpServer.sendto(bytesTServer, clientMsgAddress)
        serverMsg = udpServer.recvfrom(bufferSize)
        serverMsgMessage = serverMsg[0]
        serverMsgAddress = serverMsg[1]

        message = "Message from Client: {}".format(serverMsgMessage)
        address = "Addresss from Client: {}".format(serverMsgAddress)

        print(message)
        print(address)
        print("----------------------------------------------------------------")
    except:
        print("Not Slave")
```

❖ ***udpSlave***

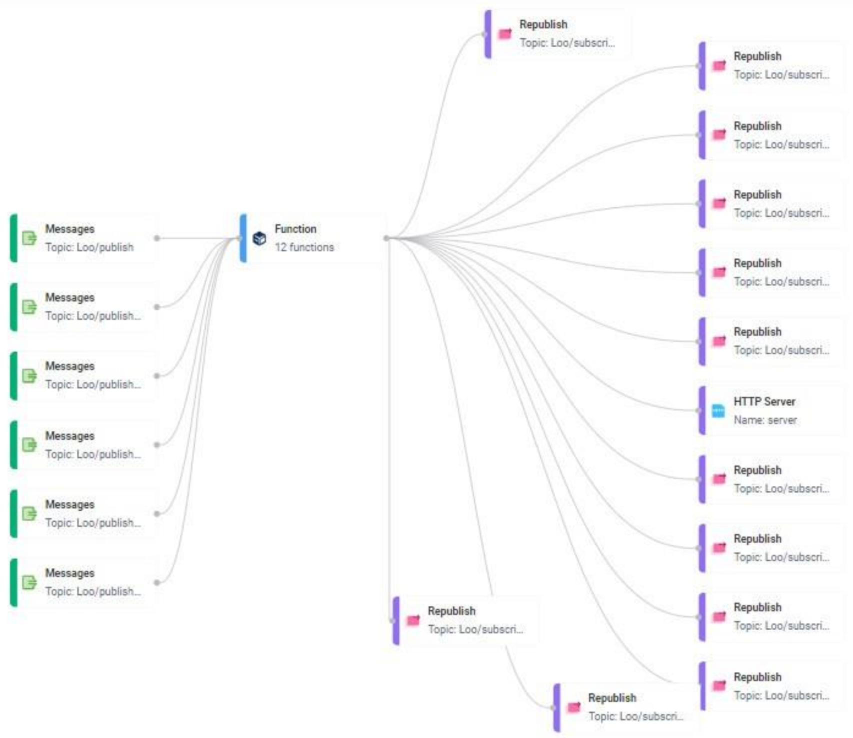
```
import socket
import ast
from grove.grove_moisture_sensor import GroveMoistureSensor
from grove.grove_ultrasonic_ranger import GroveUltrasonicRanger
from grove.grove_light_sensor_v1_2 import GroveLightSensor
from grove.display.jhd1802 import JHD1802
from rpi_ws281x import PixelStrip, Color
from gpiozero import LED

sen1 = GroveMoistureSensor(0)
sen2 = GroveUltrasonicRanger(22)
sen3 = GroveLightSensor(2)
lcd = JHD1802()
led1 = LED(5)
led2 = LED(16)

# Init UDP Server
ServerAddressPort = ("192.168.1.63", 8000)
bufferSize = 1024
udpClient = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
```

```python
# Init color text
txtReset = "\033[0m"
txtGreen = "\033[32m"

#Init data
dataRasp2 = {
    "lcd": "",
    "ledstick": 0,
    "led1": 0,
    "led2": 0
}

dataSend = {
    "moi": 0,
    "light": 0,
    "space": 0
}

# Init function
def processReceived(dataReceived):
    input_string = dataReceived.decode("utf-8")
    dataDict = ast.literal_eval(input_string)
    global dataRasp2
    dataRasp2["lcd"] = dataDict["lcd"]
    dataRasp2["ledstick"] = dataDict["ledstick"]
    dataRasp2["led1"] = dataDict["led1"]
    dataRasp2["led2"] = dataDict["led2"]
    return dataRasp2

#Xu ly RGB Ledstick
def ledstick(leds):
    LED_COUNT = 10
    LED_PIN = 18
    LED_FREQ_HZ = 800000
    LED_DMA = 10
    LED_BRIGHTNESS = 255
    LED_INVERT = False
    LED_CHANNEL = 0

    strip = PixelStrip(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA, LED_INVERT,
LED_BRIGHTNESS, LED_CHANNEL)
    strip.begin()

    R = 255 - leds
    G = leds%2
    B = 0 + leds

    for i in range(LED_COUNT):
        strip.setPixelColor(i, Color(R, G, B))
        strip.show()
 #Thuc thi
def excute():
    global dataRasp2
    lcd_str = str(dataRasp2["lcd"])
    leds = int(dataRasp2["ledstick"])
    l1 = int(dataRasp2["led1"])
    l2 = int(dataRasp2["led2"])

    lcd.clear()
    lcd.write(lcd_str)

    ledstick(leds)
    print("LCD: {}".format(lcd_str))
    print("leds: {}".format(leds))

    if l1 == 0:
        print("LED 1: OFF")
        led1.off()
    elif l1 == 1:
        print("LED 1: ON")
        led1.on()

    if l2 == 0:
        led2.off()
        print("LED 2: OFF")
    elif l2 == 1:
        led2.on()
        print("LED 2: ON")

dataid = 0
while True:
    dataid += 1
    moi = sen1.moisture
    space = round(sen2.get_distance(), 2)
    light = sen3.light
    try:
        print(txtGreen + "Tac vu gui du lieu UDP" + txtReset)
        dataSend["id"] = dataid
        dataSend["moi"] = moi
        dataSend["space"] = space
```

```python
        dataSend["light"] = light

        dataDevice = str(dataSend)
        bytesToSend = str.encode(dataDevice)

        udpClient.sendto(bytesToSend, ServerAddressPort)

        serverMsg = udpClient.recvfrom(bufferSize)
        serverMsgMessage = serverMsg[0]
        serverMsgAddress = serverMsg[1]

        message = "Message from Server: {}".format(serverMsgMessage)
        address = "addresss from Server: {}".format(serverMsgAddress)

        print(message)
        print(address)
        print("ID: ", dataid)
        print("----------------------------------------------------------------")

        print(txtGreen + "Tac vu nhan du lieu UDP" + txtReset)
        msgToClient = str("OK")
        bytesToSend = str.encode(msgToClient)

        serverMsg = udpClient.recvfrom(bufferSize)
        serverMsgMessage = serverMsg[0]
        serverMsgAddress = serverMsg[1]

        processReceived(serverMsgMessage)
        message = "Message from Server: {}".format(dataRasp2)
        address = "addresss from Server: {}".format(serverMsgAddress)

        udpClient.sendto(bytesToSend, ServerAddressPort)

        print(message)
        print(address)
        print("----------------------------------------------------------------")
        excute()
    except:
        print("Connecting Server....")
```

II. Cấu hình EMQX

❖ *Sơ đồ*



❖ *Thiết lập trong Rules*

```
FOREACH first(jq('def rem_first:
                    if length > 2 then del(.[0]) else . end;
                 def rem_last:
                    if length > 1 then del(.[-1]) else . end;
                 .id as $id|
                 .time as $time|
                 .temp as $temp |
                 .humi as $humi |
                 .relay as $relay |
                 .moi as $moi |
                 .space as $space |
                 .light as $light |
                 .ledstick as $ledstick |
                 .lcd as $lcd |
                 .led1 as $led1 |
                 .led2 as $led2 |
                 {$id, $time, $temp, $humi, $relay, $moi, $space, $light, $ledstick,
$lcd, $led1, $led2}',
            payload)) as dataFull,


        first(jq('def rem_first:
                    if length > 2 then del(.[0]) else . end;
```

```
      def rem_last:
         if length > 1 then del(.[-1]) else . end;
      .time as $time |
      $time',
    payload)) as time1,


first(jq('def rem_first:
         if length > 2 then del(.[0]) else . end;
      def rem_last:
         if length > 1 then del(.[-1]) else . end;
      .temp as $temp |
      $temp',
    payload)) as temp1,


first(jq('def rem_first:
         if length > 2 then del(.[0]) else . end;
      def rem_last:
         if length > 1 then del(.[-1]) else . end;
      .humi as $humi |
        $humi',
    payload)) as humi,

first(jq('def rem_first:
         if length > 2 then del(.[0]) else . end;
      def rem_last:
         if length > 1 then del(.[-1]) else . end;
      .relay as $relay |
        $relay',
    payload)) as relay,

first(jq('def rem_first:
         if length > 2 then del(.[0]) else . end;
      def rem_last:
         if length > 1 then del(.[-1]) else . end;
      .moi as $moi |
        $moi',
    payload)) as moi,

first(jq('def rem_first:
         if length > 2 then del(.[0]) else . end;
      def rem_last:
         if length > 1 then del(.[-1]) else . end;
      .light as $light |
        $light',
    payload)) as light,

first(jq('def rem_first:
         if length > 2 then del(.[0]) else . end;
      def rem_last:
         if length > 1 then del(.[-1]) else . end;
```

```
            .lcd as $lcd |
              $lcd',
          payload)) as lcd,

        first(jq('def rem_first:
                 if length > 2 then del(.[0]) else . end;
              def rem_last:
                 if length > 1 then del(.[-1]) else . end;
              .space as $space |
                $space',
            payload)) as space1,

        first(jq('def rem_first:
                 if length > 2 then del(.[0]) else . end;
              def rem_last:
           if length > 1 then del(.[-1]) else . end;
              .ledstick as $ledstick |
                $ledstick',
            payload)) as ledstick,

        first(jq('def rem_first:
                 if length > 2 then del(.[0]) else . end;
              def rem_last:
                 if length > 1 then del(.[-1]) else . end;
              .led1 as $led1 |
                $led1',
            payload)) as led1,

        jq('def rem_first:
                 if length > 2 then del(.[0]) else . end;
              def rem_last:
                 if length > 1 then del(.[-1]) else . end;
              .led2 as $led2 |
                $led2',
            payload) as led2

FROM
  "Loo/publish",
  "Loo/publish/relay",
  "Loo/publish/led1",
  "Loo/publish/led2",
  "Loo/publish/ledstick",
  "Loo/publish/lcd"
```

III. Video minh chứng

Link: https://youtu.be/c8B5tFdFy8I?si=eCdYmwzjv-BYCefx