IS4151/IS5451 – AIoT Solutions and Development

AY 2024/25 Semester 2

Practical Lab 05 – IoT Backend Integration

Postman

Practical -> Stage your own backend
-> real hardware set up or postman

Part 1 – Basic Programming

PE05-1-1 – Cloud-enabled Weather Station

Recall that in PE03-2-1, you have created a simple home weather station. The weather station consists of a temperature indicator gauge and a humidity indicator gauge for monitoring the ambient temperature and humidity level using a Raspberry Pi and the Bosch BME280 environmental sensor. We will now extend its capability with an IoT backend.

Design and create a RESTful web service endpoint on a cloud server to record the ambient temperature detected by the Raspberry Pi based on the following rules. Remember to take into consideration the threshold for the variance in the temperature value.

a. On start-up, the Raspberry Pi should report the initial temperature to the cloud server.

b. Whenever the temperature changes (measured at one minute interval), i.e., the current temperature is lower/higher than the previous temperature, report the current temperature to the cloud server.

c. Whenever the user resets the baseline temperature, report the new baseline temperature to the cloud server.

d. On shutdown, the Raspberry Pi should report the final temperature to the cloud server.

All temperature values should be reported to the cloud server together with the current timestamp. A sample event table is provided below:

Time (in Minute)	Temperature	LED	Event	Report Event	Remark
0	28 (start-up)	Green	Start-up	Yes	Initial Temp.
1	28	Green		No	
2	29	Red	Change	Yes	Current Temp.
3	29	Red		No	
4	29	Red		No	
5	29 (reset)	Green	Reset	Yes	Baseline Temp.
6	29	Green		No	
7	28	Blue	Change	Yes	Current Temp.
8	28 (shutdown)	Blue	Shutdown	Yes	Final Temp.

Essentially, each data record that is transmitted to the cloud server will include the timestamp, event type and temperature reading.

need change w8 src case study -> py and yml
- include need event

Tan Wee Kek, tanwk@comp.nus.edu.sg

if no change then no report -> no need waste space lol
-> T4 - T6 if no change no need lol
But ultimately depends on usecase

get another endpoint to support humidity
or change the same endpoint -> to handle
temperature or humidity

Repeat the above process by designing and creating a RESTful web service endpoint on a cloud server to record the ambient humidity level detected by the Raspberry Pi based on the same set of rules above.

After you have completed the development of the RESTful web services on the cloud server, perform the following tasks:

dashboard lah, average daily humidity ovr
time period Information Services

a. What are some information services that you can provide from the Raspberry Pi itself (acting as the fog processor) and your laptop (acting as the cloud server)?

b. Implement these information services to create a useful weather station IoT system.

c. Recall that you have more than one Raspberry Pi. Set up a second Raspberry Pi to act as a second weather station. How would the addition of new weather station(s) change your IoT system's set up process and the provision of information services?

PE05-1-2 – Cloud-enabled Smart Lighting Switch

Recall that in PE04-1-3 and PE04-2-1, you have created a smart lighting switch. More specifically, the micro:bit device interacts with a Raspberry Pi device over its Bluetooth UART service to enable the micro:bit device to act as a wireless remote control switch for a hypothetical smart light.

Design and create RESTful web service endpoints on a cloud server to record the various events that are generated by one or more smart lighting switches:

- Switch Off the Light
- Switch On the Light
- Dim the Light to 50% Brightness
- Restore the Light to Full Brightness

only event -> no need parameter
values -> dont need to know how
bringt just know 50%

After you have completed the development of the RESTful web service endpoints on the cloud server, perform the following tasks:

user habits -> energy consumption, rule based protocol
-> simple mechanism, turn off or turn on preferences

a. What are some information services that you can provide from the Raspberry Pi itself (acting as the fog processor) and your laptop (acting as the cloud server)?

b. Implement these information services to create a useful smart lighting switch IoT system.

c. Recall that you have more than one Raspberry Pi. Set up a second Raspberry Pi to act as a second smart lighting switch (including an actual red colour LED). How would the addition of new smart lighting switch(es) change your IoT system's set up process and the provision of information services?

Part 2 – Intermediate Programming

PE05-2-1 – Cloud-enabled Forest Fire Detection and Warning System

Recall that in Lecture 08, we had discussed about the **Ambient Temperature Case Study**. One of the real-world use cases for such an IoT system is to monitor the ambient temperature of forested areas and use the temperature values as the basis to detect the outbreak of forest fires. Advanced use cases of the temperature values may include predicting the size and path of a forest fire, etc.

The sample code given to you in the lecture involves one-way movement of data from the node devices to the integrated hub and fog processor and then to the cloud server. More specifically, the node measures the ambient temperature whenever instructed by the hub to do so. The fog processor then collects the temperature values and stored them on a local database before relaying to the cloud server at a periodic interval.

In this exercise, you are required to convert the sample setup demonstrated in the lecture into a two-way or bidirectional IoT system based on the following scenario:

- The cloud server continuously monitors the ambient temperature values across the entire geographical region.
- Upon detecting a hot spot, i.e., a district with ambient temperature greater than 50 degrees Celsius, the cloud server will instruct the affected fog processor to issue an evacuation alarm to all citizens within the geographical region, i.e., all sensor nodes in districts within the affected area.
- The evacuation alarm will manifest as a red LED that is attached to the Raspberry Pi blinking continuously, and the 5 x 5 LED of the micro:bit devices blinking continuously.
- The flow of the command to trigger the evacuation alarm is from the cloud server to the affected integrated hub and fog processor and then to the affected nodes.

Design a suitable hardware and software setup to implement the forest fire detection and warning system.

Hint: To implement such a system, you would need two Python programs running on the Raspberry Pi, i.e., one acting as the web service and another acting as the control application. Two Python programs running on the same host can communicate with each other using sockets.

The following code fragment demonstrates how to run a server program listening on port 8888:

concurrent server

IPC -> socket programming

```
import socket

def server():
      host = socket.gethostname()
      port = 8888
```

if need to run the raspberry pi then is like another server so its like waste energy

```
        s = socket.socket()
        s.bind((host, port))

        s.listen(1)
        client_socket, adress = s.accept()

        print("Connection from: " + str(adress))

        while True:
            data = client_socket.recv(1024).decode('utf-8')

            if not data:
                break

            print('From online user: ' + data)
            data = data.upper()
            client_socket.send(data.encode('utf-8'))

        client_socket.close()

if __name__ == '__main__':
    server()
```

The following code fragment demonstrates how to run a client program communicating with the server program on port 8888:

```
import socket

def client():
    host = socket.gethostname()
    port = 8888

    s = socket.socket()
    s.connect((host, port))

    message = input('-> ')

    while message != 'q':
        s.send(message.encode('utf-8'))
        data = s.recv(1024).decode('utf-8')
        print('Received from server: ' + data)
        message = input('==> ')

    s.close()

if __name__ == '__main__':
    client()
```

The preceding code fragment demonstrates a single instance server. You can convert it into a concurrent server in Python using multithreading to service multiple clients at the same time.

```
import socket
import _thread as thread

def service_client(client_socket, address):

      print("Connection from: " + str(address))

      while True:
            data = client_socket.recv(1024).decode('utf-8')

            if not data:
                  break

            print('From online user: ' + data)
            data = data.upper()
            client_socket.send(data.encode('utf-8'))

      client_socket.close()

def server():

      try:

            host = socket.gethostname()
            port = 8888

            s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
            s.setsockopt(socket.SOL_SOCKET,
                  socket.SO_REUSEADDR, 1)
            s.bind((host, port))

            while True:

                  s.listen(1)
                  client_socket, address = s.accept()
                  # s.setblocking(False)
                  thread.start_new_thread(service_client,
                     (client_socket, address))

      except KeyboardInterrupt:

            print('Program terminated!')
```

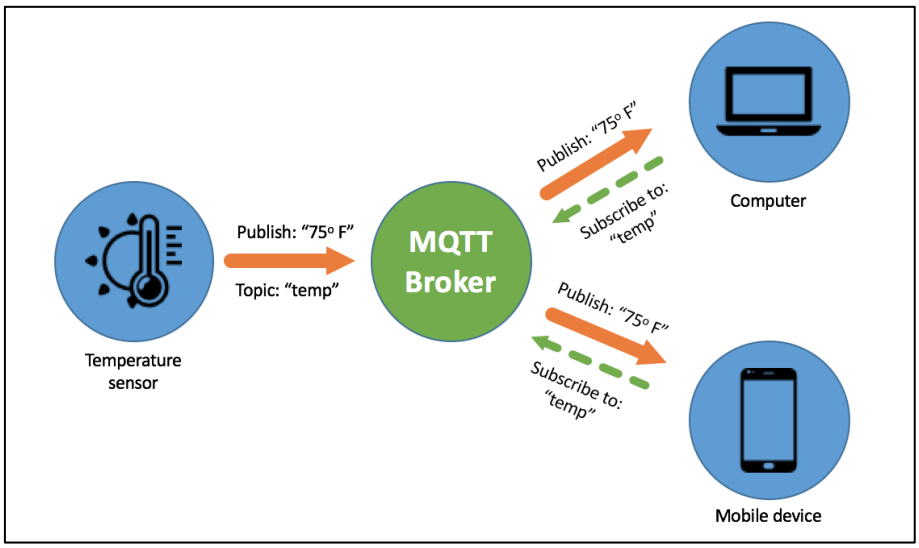
```
        finally:
            s.close()
if __name__ == '__main__':
    server()
```

Part 3 – Recitation on MQTT

MQTT (Message Queuing Telemetry Transport) is an open OASIS and ISO standard lightweight, publish-subscribe network protocol that transports messages between devices. The protocol was invented in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link) to facilitate connection with oil pipelines via satellite with minimal battery loss and minimal bandwidth. In March 2019, OASIS ratified the new MQTT 5 specification. This new version introduced several new features that are required for IoT applications deployed on cloud platforms, and those that require more reliability and error handling to implement mission-critical messaging.

The MQTT publish-subscribe model decouples the client that sends a message (the publisher) from the client or clients that receive the messages (the subscribers). MQTT uses the topic (subject) of the message to determine which message goes to which subscriber. A topic is a hierarchically structured string resembling a URI that can be used to filter and route messages. The counterpart of the MQTT client is the MQTT broker. The broker is responsible for receiving all messages, filtering the messages, determining who is subscribed to each message, and sending the message to these subscribed clients.

The diagram below shows the schematic overview of the MQTT protocol:



In Python, we can create MQTT publisher and subscriber clients easily using the **Eclipse Paho MQTT Python client library** (https://pypi.org/project/paho-mqtt/). The sample code can be downloaded from the Canvas Files tool. This demonstration uses the free public MQTT broker provided by EMQ X. A public broker such as this does not provide privacy protection. As such, it should only be used for development purpose. For production, you can

consider using **Eclipse Mosquitto** (https://mosquitto.org/), which is an open source MQTT broker.

PE05-3-1 – Cloud-enabled Forest Fire Detection and Warning System with MQTT

The cloud-enabled forest fire detection and warning system in PE05-2-1 currently uses RESTful web services to send commands to the integrated hub and fog processor. The RESTful web services on the integrated hub and fog processor then need to forward the commands to node devices via BLE or radio (with serial USB).

Convert your solution for PE05-2-1 to use the MQTT protocol instead of RESTful web services. Document the changes that you have made to the hub program.

Which approach would you prefer, RESTful web services or MQTT?

SMART ACTUATION !!!!

NEED MQTT WE NEED TO TRY OUT