Eugenio Principi - s236654
Arya Houshmand - s247275
Leonardo Perugini - s249807

Il primo esercizio del laboratorio software 3 non viene riportato in quanto è un esercizio che serve a prendere dimestichezza con MQTT, i client Mosquitto ed in generale il paradigma di comunicazione Publish / Subscribe.

In generale per tutti gli esercizi di questo laboratorio software si fa spesso riferimento agli esercizi precedentemente svolti nei laboratori software e hardware, in particolare quelli relativi al laboratorio SW2 e HW3: quindi spesso saranno riportati file che sono stati sviluppati la prima volta per altri laboratori software e modificati secondo le esigenze.

Esercizio 3.2

Nell'esercizio 3.2 viene sviluppato un *subscriber MQTT* per ricevere i valori di temperatura misurati da un sensore collegato alla scheda Arduino facendo riferimento al laboratorio HW3.
Il *message broker*, la relativa *porta* da utilizzare e gli *end-points* sono informazioni che vengono dedotte dal *Catalog* che viene passato come parametro all'interno del costruttore del *subscriber*.
Infine nella inizializzazione del *subscriber MQTT* vi è anche la registrazione al *Catalog* come nuovo servizio.

```
def __init__(self, clientID, topic, broker, catalog):
    self.catalog = catalog
    self.clientID = clientID
    # create an instance of paho.mqtt.client
    self._paho_mqtt = PahoMQTT.Client(clientID, False)

    # register the callback
    self._paho_mqtt.on_connect = self.myOnConnect
    self._paho_mqtt.on_message = self.myOnMessageReceived
    # l'ip del catalog corrisponde al broker utilizzato per MQTT
    self.messageBroker = catalog.ip
    self.port = catalog.port
    # recupera i topic del device arduino precedentemente registrato, cont
    # al momento della registrazione
    self.topic = []
    for var in catalog.getDevice('Yun').get('endpoints'):
        self.topic.append(var)

    self.temperature = 0

    #registrazione come nuovo servizio del catalog
    service_content = {
        "service_id": "MQTT_temp_subscriber",
        "description": "temperature subscriber mqtt service",
        "endpoints": "/tiot/9/temperature"
    }
    self.catalog.addService(service_content)
```

In *Esercizio2.py* è contenuto il main, insieme alla classe *REST* ed il *Client* che sono stati presi dal laboratorio precedente. Prima di tutto impostiamo le configurazioni standard di CherryPy e dopodiché viene creato il *Subscriber MQTT.*

```
cherrypy.engine.start()
c = Client(infoDevice, 'http://127.0.0.1:8080/addDevice')

# boot del subscriber mqtt
sub = myMQTTSubscriber("subscriber_temp", "", "", catalog)
sub.start()

cherrypy.engine.block()
```

Esercizio 3.3

Il terzo esercizio è piuttosto simile alla consegna precedente, ma questa volta viene sviluppato un *publisher MQTT* che invii dei comandi alla scheda Arduino per spegnere o accendere il LED. Per quanto riguarda il codice lato Arduino, come prima, si fa riferimento al codice scritto per il laboratorio HW3, del quale non è stato cambiato niente.
Nel file *pub.py* abbiamo sviluppato un semplice *publisher MQTT*, il quale al suo avvio si registra al *Catalog* come nuovo servizio e reperisce le informazioni di broker e port, proprio come nell'esercizio precedente.

In *Esercizio3.py* invece viene lanciato il programma vero e proprio: rifacendoci all'esercizio 3 del laboratorio SW2, abbiamo riportato il *Client* per registrare un nuovo dispositivo, e la classe *REST* per gestire il *Catalog*. A questo punto nel main, dopo aver impostato le configurazioni relative a CherryPy e registrato il dispositivo, creiamo un istanza del *Publisher* e inviamo i comandi di attuazione alla scheda Arduino nel formato SenML.

```
p = myMQTTpub("Pub_Led", catalog)
p.start()


while True:
    print("Inserisci 1 per accendere il Led, 0 per spegnere:")
    val = input()
    val = int(val)

    ledCommands = {
        "bn": "Yun",
        "e": [{"n": "led", "t": 'null', "v": val, "u": 'null'}]
    }
    p.myPublish("/tiot/9/led", json.dumps(ledCommands))
    time.sleep(60)
```

Esercizio 3.4

In quest'ultimo esercizio era richiesto di modificare lo *Smart Home Controller* sviluppato precedentemente nel laboratorio HW2 in versione *Remote*, ovvero in grado di inviare e ricevere informazioni e comandi di attuazione grazie alla comunicazione di tipo *Publish* / *Subscribe* con *MQTT*.

Dal momento che, aggiungendo tutto il necessario per far sì che questa versione del programma possa lavorare in remoto, la memoria della scheda Arduino quasi si esaurisce e quindi viene mostrato un messaggio di warning in cui si dice che la stabilità della scheda potrebbe essere compromessa.
Infatti facendo varie prove a volte il codice funzionava e altre no, oppure il *loop()* eseguiva solo uno / due cicli e poi si bloccava; questo è probabilmente dovuto al fatto che la Arduino non era stabile.

La versione sviluppata nel laboratorio HW2 richiedeva chiaramente meno memoria e si inserivano i comandi tramite porta seriale, rendendo tutto il codice più stabile facendo in modo che la *Yùn* non si bloccasse.

Tuttavia lo *Smart Home Controller* remoto sarebbe molto più comodo per un utente, poiché questo non dovrà comunicare tramite porta seriale con la scheda e quindi essere fisicamente accanto ad essa, ma potrebbe interagire comunque con la scheda trovandosi in un luogo differente e connesso ad un'altra rete.