Laboratorio SW 2 **Tecnologie per IoT** **Gruppo 9**

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

Esercizio 1

Innanzitutto abbiamo creato in tre moduli differenti le classi che rappresentano gli *users*, i *devices* e i *services* con i relativi dati che questi necessitano (qui sotto riportiamo solamente la classe User).

```
class User():

    def __init__(self, content):

        self.user_id = content["user_id"]
        self.name = content["name"]
        self.surname = content["surname"]
        self.email = content["email"]
```

In un modulo a parte invece si è creato come classe il *Catalog*, contente le informazioni del Catalog e i metodi necessari, per esempio, a registrare o fornire gli utenti. In questa classe è inoltre presente il metodo *__refreshDevices*, che si occupa di eliminare i device che hanno un timestamp più vecchio di 2 minuti.

```
    def __refreshDevices(self):
        while True:
            self.clearOldDevices()
            time.sleep(60)

    def clearOldDevices(self):
        min_time = int(time.time()) - 120
        for k in list(self.devices.keys()):
            if (self.devices[k]).timestamp < min_time:
                del self.devices[k]
                print('Device %s eliminato.' % k)
```

Nel file *Esercizio1.py* invece è contenuta l'applicazione REST che gestisce il Catalog, creata utilizzando il framework CherryPy. In particolare abbiamo implementato i metodi GET e POST: GET nel caso si volessero ricevere determinate informazioni (avere i vari devices, utenti, etc.) e POST nel caso si volessero registrare nuovi utenti, servizi o dispositivi.
Qui di seguito riportiamo il metodo GET

```python
@cherrypy.expose
def GET(self, *uri, **params):
    if len(uri) == 1:
        if uri[0] == 'getInfo':
            r = self.catalog.getInfo()
        elif uri[0] == 'getDevices':
            r = self.catalog.getDevices()
        elif uri[0] == 'getUsers':
            r = self.catalog.getUsers()
        elif uri[0] == 'getServices':
            r = self.catalog.getServices()
        else:
            raise cherrypy.HTTPError(404, "unknown URI.")
    elif len(uri) == 2:
        if uri[0] == 'getDevice':
            r = self.catalog.getDevice(uri[1])
        elif uri[0] == 'getUser':
            r = self.catalog.getUser(uri[1])
        elif uri[0] == 'getService':
            r = self.catalog.getService(uri[1])
        else:
            raise cherrypy.HTTPError(404, "unknown URI.")

        if r is None:
            cherrypy.response.status = 404
    else:
        raise cherrypy.HTTPError(404, "unknown URI.")
    if r == "":
        raise cherrypy.HTTPError(404, "Can't satisfy the request.")
    else:
        return r
```

Esercizio 2

In questo esercizio viene sviluppato un *Client* per effettuare richieste al Catalog sviluppato secondo le architetture REST nell'esercizio precedente.
La classe Client implementa i metodi necessari a richiedere informazioni dal Catalog (device, user, etc.).

```
class Client:

    def getBrokerInfo(self):
        r = requests.get('http://localhost:8080/getInfo')
        info = r.json()
        return info

    def getDevices(self):
        r = requests.get('http://localhost:8080/getDevices')
        devices = r.json()
        return devices

    def getDevice(self, device_id):
        r = requests.get('http://localhost:8080/getDevice/' + device_id)
        device = r.json()
        return device

    def getUsers(self):
        r = requests.get('http://localhost:8080/getUsers')
        users = r.json()
        return users

    def getUser(self, user_id):
        r = requests.get('http://localhost:8080/getUser/' + user_id)
        device = r.json()
        return device
```

Esercizio 3

In *Esercizio3.py* vi è un Client che emula un dispositivo IoT registrandosi o aggiornando vecchie informazioni nel Catalog ogni minuto. Per fare ciò abbiamo implementato i due metodi relativi nel seguente modo:

```
class Client:

    def __init__(self, info_device, registration_endpoint):
        self.device = info_device
        self.registration_endpoint = registration_endpoint
        threading.Thread(target=self.__refreshRegistration).start()

    def register(self):
        r = requests.post(self.registration_endpoint, json=self.device)

    def __refreshRegistration(self):
        while True:
            self.register()
            time.sleep(60)
```

Esercizio 4

Nel quarto esercizio abbiamo ripreso il codice lato Arduino relativo all'esercizio 3.2 del laboratorio hardware 3. Nell'encoding SenMl abbiamo modificato il valore del base name, che viene utilizzato dal Catalog come id per il device, e quindi abbiamo inserito come valore relativo a "bn" il nome "ArduinoYun". Le richieste HTTP POST vengono inviate tramite *curl* dalla scheda Arduino.

Per quanto riguarda invece l'estensione del primo esercizio di questo laboratorio, abbiamo aggiunto nella gestione delle richieste POST una nuova uri, nel caso si volesse aggiornare il valore di temperatura, e nel Catalog una nuova funzione, per registrare il device o effettuarne il refresh, che riportiamo qui sotto:

```
def refreshTemperature(self, content):
    deviceID = content['bn']
    d = self.getDevice(deviceID)
    if d!=None:
        d.timestamp = int(time.time())
    else:
        #aggiorna il contenuto di content
        content['device_id'] = deviceID
        content['resources'] = "sensore di temperatura"
        content['endpoints'] = "http://192.168.1.1:8080/refreshTemperature"
        self.addDevice(content)
```

Esercizio 5

Per estendere le funzionalità del Catalog in modo tale da adattarsi a MQTT abbiamo creato un servizio generico MQTT che esegue la sottoscrizione ad un topic nel file *myqtt.py*, il quale contiene l'implementazione dei metodi *start, stop, etc.*
In *Esercizio5.py* sono implementati i metodi GET e POST in maniera analoga al primo esercizio, mentre nel *main* viene creato anche il servizio MQTT.

```
if __name__ == "__main__":
    catalog = Catalog('mqtt.eclipse.org', '1883')
    mymqtt = myMQTT('MainServer', 'test/topic', 'mqtt.eclipse.org', catalog)
    threading.Thread(target=mymqtt.start, daemon=True).start()
    # Standard configuration to serve the url "localhost:8080"
    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tool.session.on': True
        }
    }
    cherrypy.quickstart(Rest(catalog), '/', conf)
```

Esercizio 6

L'ultimo esercizio presenta lo sviluppo di un *publisher* MQTT che ogni minuto registra nel Catalog un nuovo dispositivo o ne aggiorna le informazioni.
Abbiamo dunque creato un Client che funziona da publisher.
Qui di seguito alcuni dei metodi presenti nella classe *myPublisher()*:

```
def myPublish(self, topic, message):
    # publish a message with a certain topic
    self._paho_mqtt.publish(topic, message)

def onPublish(mosq, obj, mid):
    print("Comando inviato con message id: " + str(mid))
```