POLITECNICO DI TORINO

Alberto Simone s247818

Balduzzi Luca s248398

Canensi Emanuele s235263

Corso di Tecnologie per IoT

01UEIOA

Relazione Laboratorio Software N°2

Scopo del laboratorio: familiarizzare con l'ambiente IoT realizzando un Catalog e alcuni client di prova.

Esercizio 1

Nel primo esercizio si procede con la progettazione di un Catalog per la gestione di un sistema IoT.

Si è scelto di creare una classe per ogni componente che il Catalog deve gestire: utenti, dispositivi, message broker, servizi e mqtt.

Il Catalog espone i servizi REST tramite CherryPy.

Descrizione ENDPOINT

```
##
# Message Broker
#
# GET
# - /messagebroker          Retrieve all the registered devices
# POST
# - /messagebroker/new      Add a new message broker
##


##
```

```
# Devices
#
# GET
# - /devices                Retrieve all the registered devices
# - /devices/:deviceId      Retrieve a specific device with a deviceID
# POST
# - /devices/new            Add a new device
##


##
# Users
#
# GET
# - /users                Retrieve all the registered users
# - /users/:userID        Retrieve a specific user with a userID
# POST
# - /users/new            Add a new user
##


##
# Services
#
# GET
# - /services
# - /services/:serviceID
# POST
# - /services/new
##
```

```
##
# DEVICES DATA
# - /temperature
# - /avgtemperature
# - /heating
# - /cooling
# - /presence
# - /lighting
##
```

Ogni classe contiene al suo interno una lista di tutti i componenti appartenenti ad esse in sincronismo con quanto viene salvato su un file JSON proprio per ogni classe; il sincronismo si ha quando il programma è in funzione; il Catalog si sincronizza con i database JSON ogni volta che viene avviato.

Classe per la gestione dei Devices

Formato json:

```
{
    deviceID: "",
    rest: "",
    mqtt: ""
    resources: [],
    timestamp: ""
}
```

Quando un device si connette per la prima volta al Catalog viene registrato con i suoi parametri, attraverso una richiesta POST all'endpoint /devices/news, e gli viene assegnato un ID; una volta che è stato registrato il dispositivo inizierà a trasmettere le risorse. Ogni sensore rappresenta un device diverso, la scheda quindi gestisce diverse comunicazioni per ogni sensore ed ognuno comunica verso il catalog

con un topic ben distinto in MQTT. Il Catalog assegna inoltre il timestamp. Il device manager ha anche il compito di catalogare tutte le letture fornite da ogni sensore/device in una lista di risorse.

All'interno della classe si trovano anche le funzioni per ottenere la lista dei dispositivi registrati e il singolo dispositivo.

Classe per la gestione del Message Broker

Formato json:

```
{
    "url": "",
    "port": "",
    "catalogTopic": ""
}
```

Quando si avvia il Catalog questo apre il file json del message broker, mb.json, per salvare i dati relativi alla connessione da stabilire e al topic al quale iscriversi.

Classe per la gestione di MQTT

All'interno di questa classe sono contenute tutte le istruzioni per permettere al Catalog di stabilire una connessione mqtt attraverso le informazioni del message broker; si trovano inoltre le funzioni che permettono la gestione delle funzionalità publish e notify.

Classe per la gestione degli User

Formato json:

```
{
    userID: "",
    name: "",
    surname: "",
    email: ""
}
```

Un utente per registrarsi deve seguire l'endpoint /user/new ed effettuare una richiesta di tipo POST passando come parametri il nome, cognome e l'indirizzo e-mail; il dato userID gli viene assegnato dall Catalog che si occupa di tenere traccia di tutti gli utenti registrati.

All'interno della classe si trovano anche le funzioni per ottenere la lista degli utenti registrati e il singolo utente.

Classe per la gestione dei Services

Formato json:

```
{
    serviceID: "",
    description: "",
    rest: "",
    mqtt: "",
    timestamp: ""
}
```

Un servizio per registrarsi deve seguire l'endpoint /services/new ed effettuare una richiesta di tipo POST; il dato serviceID gli viene assegnato dall Catalog che si occupa di tenere traccia di tutti i servizi registrati. Viene anche memorizzata la tipologia di connessione del servizio; il Catalog assegna inoltre il timestamp.

All'interno della classe si trovano anche le funzioni per ottenere la lista dei servizi registrati e il singolo servizio.

Esercizio 2

Nel secondo esercizio si procede con lo sviluppo di un client avente la funzione di testare i servizi di tipo REST sviluppati all'interno del Catalog.

Si accede alle varie funzionalità tramite input da tastiera.

Il client effettua richieste REST usando il modulo python *requests*.

Esercizio 3

Nel terzo esercizio si procede con lo sviluppo di un client avente la funzione di emulare un dispositivo IoT.

Il client deve registrarsi al Catalog e periodicamente effettuare richieste REST per l'update del timestamp (effettuato a livello Catalog).

Si utilizza la libreria *time* all'interno di un ciclo while per simulare lo stato di idle di un dispositivo reale.

Esercizio 4

Nel quarto esercizio si procede con lo sviluppo di un client avente la funzione di emulare un dispositivo IoT.

Il client deve registrarsi al Catalog e periodicamente effettuare richieste REST per l'update della temperatura rilevata e il corrispettivo timestamp (quest'ultimo effettuato a livello Catalog).

Si utilizza la libreria *time* all'interno di un ciclo while per simulare lo stato di idle di un dispositivo reale.

Esercizio 5

Nel quinti esercizio si vanno ad estendere le funzionalità del Catalog implementando la libreria Paho per permettergli di gestire richieste mqtt.

L'esercizio è stato sviluppato all'interno della classe MQTT all'interno dell'esercizio 1.

Esercizio 6

Nel sesto esercizio si procede con lo sviluppo di un client avente la funzione di emulare un dispositivo IoT.

Il client deve registrarsi al Catalog e periodicamente pubblicare messaggi mqtt per l'update del timestamp (effettuato a livello Catalog).