Philips Hue

Esempio di interazione con lampadine Philips Hue

Sistema di home automation: Bridge + lampadine/striscia LED

Il bridge è collegato alla LAN di casa, ha un suo indirizzo IP, ed espone una interfaccia REST

Il bridge forma una PAN con lampadine o strisce LED utilizzando il protocollo Zigbee (profilo Smart Lighting), un protocollo wireless a basso consumo, aperto.

Bridge e lampadine

- Le chiamate alle API REST del bridge possono essere usate per leggere lo stato delle lampadine, o per modificarlo. Il bridge comunicherà con la lampadina via Zigbee per impostare lo stato.
- Si può accedere al bridge tramite un'app della Philips (per potersi collegare occorre essere fisicamente vicino al bridge perché per associare il cellulare al bridge occorre premere un pulsante sul bridge stesso) oppure si può sviluppare un client che effettua le chiamate RESTful al bridge. Tipicamente il bridge non si espone su internet, ma viene visto solo nella intranet.

TUTTE LE API DEL BRIDGE SI POSSONO TROVARE QUI (Nota: occorre registrarsi):

https://developers.meethue.com/develop/hue-api/

L'emulatore

Esiste un emulatore che implementa (parte del)le funzionalità disponibili anche sul sistema reale. Per scaricare il jar:

http://steveyo.github.io/Hue-Emulator/

- Una volta scaricato il jar dell'emulatore lo si può avviare
- Dalla GUI scegliere la porta su cui offre il servizio e premere Start

(Nota: si può verificare con CURL che risponda alle richieste)

```
curl localhost:8000/api/newdeveloper/lights
```

RESTful API

URL per accedere al server (emulatore o bridge)

http://<IP SERVER>:8000/api/<USERNAME>/lights

http://<IP SERVER>:8000/api/<USERNAME>/lights/<N>

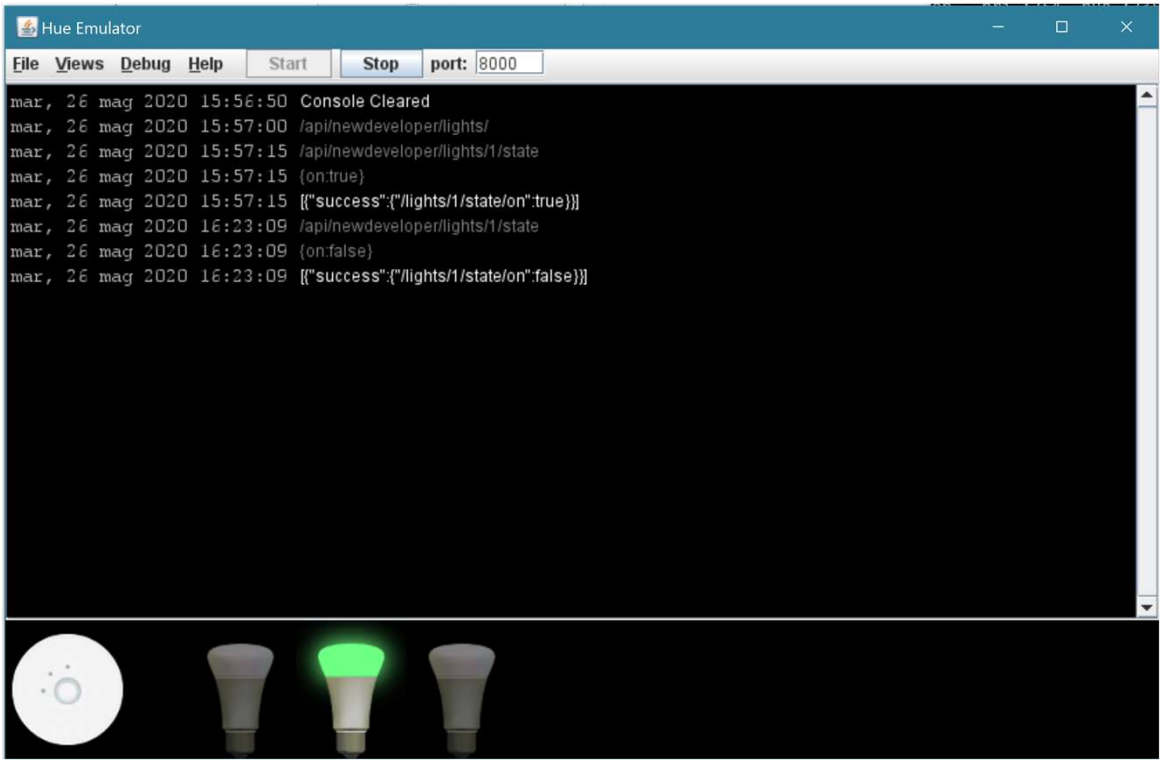
http://<IP SERVER>:8000/api/<USERNAME>/lights/<N>/state

Dove:

<IP SERVER> è l'indirizzo IP dell'emulatore (localhost) o del bridge

<USERNAME> è "newdeveloper" per l'emulatore mentre per il bridge è una stringa lunga e complicata che si può generare tramite la procedura spiegata nella Sezione «Getting Started» del sito per Hue Developer

Emulatore per test RESTful API



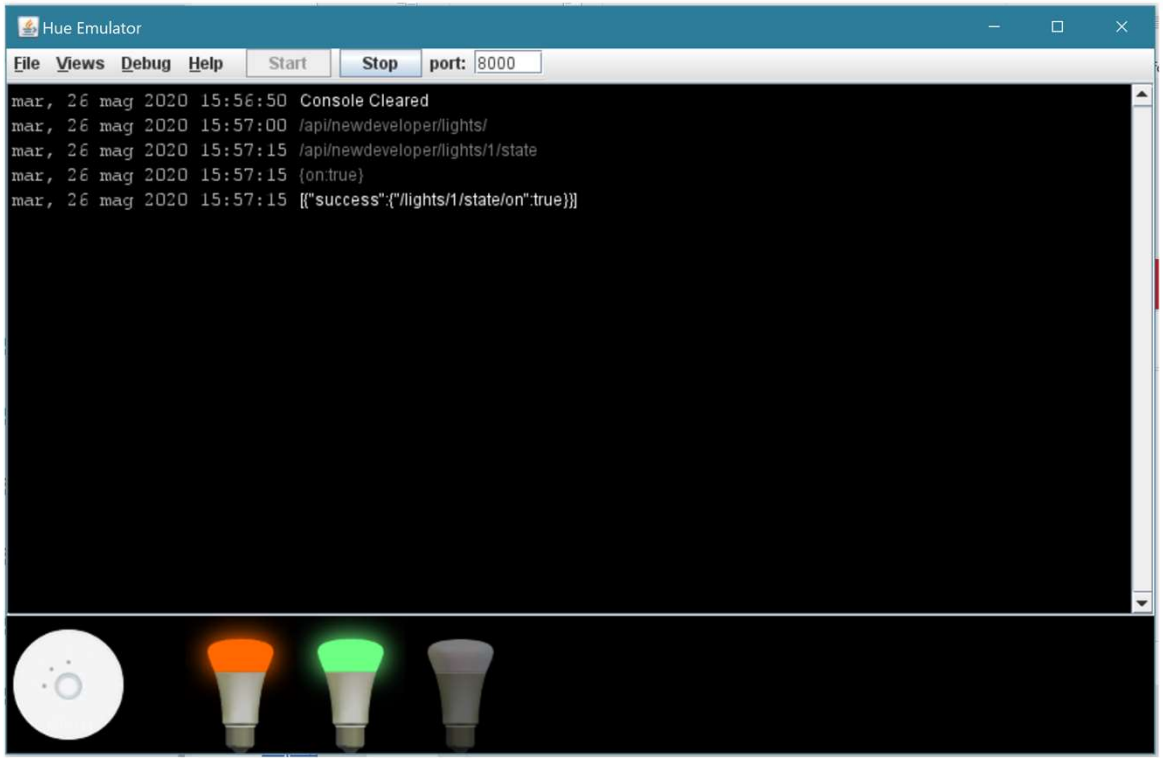
curl http://localhost:8000/api/newdeveloper/lights/

Legge (GET) l'elenco delle lampadine presenti e di ciascuna descrive le caratteristiche tecniche e lo stato.

VALORE RESTITUITO DALLA GET


```
{"1":{"modelid":"LCT001",
      "name":"Hue Lamp 1",
       "swversion":"65003148",
"state":{"xy":[0,0],"ct":0,"alert":
"none","sat":254,"effect":"none","b
ri":254,"hue":4444,"colormode":"hs"
,"reachable":true,"on":false},
"type":"Extended color light»,
"pointsymbol":{"1":"none","2":"none
","3":"none","4":"none","5":"none",
"6":"none","7":"none","8":"none"},"
uniqueid":"00:17:88:01:00:d4:12:08-
0a"},
"2":{"modelid":"LCT001","name":"Hue
Lamp 2","swversion":"65003148",
"state":{"xy":[0.346,0.3568],"ct":2
01,"alert":"none","sat":144,"effect
":"none","bri":254,"hue":23536,"col
ormode":"hs","reachable":true,"on":
true},"type":"Extended color
light", ... }
```

Accendere una lampadina




```
$ curl -X PUT -d  "{on:true}"
http://localhost:8000/api/newdeveloper/lights/1/state
```

>>> RISPOSTA [{"success":{"/lights/1/state/on":true}}]

Vedere: https://developers.meethue.com/develop/hue-api/lights-api/#set-light-state

Impostare i colori: HSL

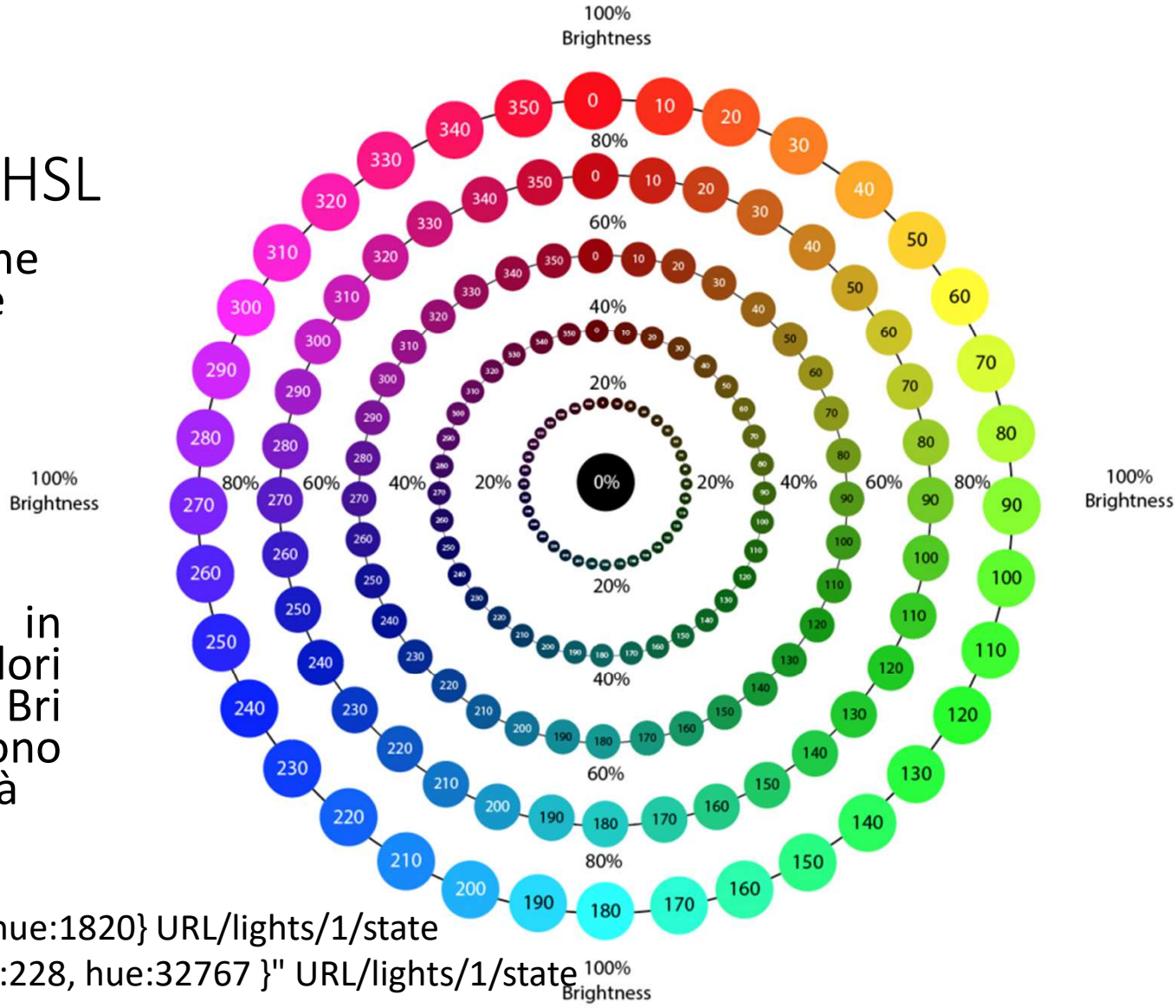
Nello stato delle lampadine si possono impostare i tre parametri:

hue (0:65535)

sat (0:254)

bri (1:254)

Facendo le proporzioni in base alla ruota dei colori (Hue in gradi 0:360; Sat e Bri in percentuale) si possono ottenere le diversi tonalità



Esempi:

ROSA curl -X PUT -d "{bri:228, sat:80, hue:1820} URL/lights/1/state

TURCHESE curl -X PUT -d "{bri:152, sat:228, hue:32767 }" URL/lights/1/state

Un semplice client che interagisce con l'emulatore

- Vediamo ora un semplice client che interagisce con l'emulatore delle Philips Hue. Se utilizzato in laboratorio può funzionare con le lampadine fisiche semplicemente sostituendo l'URL con l'indirizzo IP del bridge e lo <username> con una stringa ottenuta direttamente dal Bridge con una opportuna procedura da effettuare in loco.

Spring-web framework + GSON

- Per questo esempio utilizziamo una parte del framework Spring che permette di fare chiamate REST (classe RestTemplate). Quest'ultimo utilizza GSON come libreria per gestire la serializzazione e deserializzazione di oggetti Java in JSON.

INSERIRE IN build.gradle:

```
implementation 'org.springframework:spring-web:5.1.5.RELEASE'
implementation 'com.google.code.gson:gson:2.8.5'
```

La classe EsperimentiHue

- La classe Hue contiene il main() e utilizza le seguenti classi e metodi della libreria spring-web:
- Classe RestTemplate con i metodi getForObject, put
- Classe HttpHeaders con il metodo getContentType
- Classe HttpEntity

Si veda anche: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html

Import

```
import java.util.HashMap;
import java.util.Map;

import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.web.client.RestTemplate;
```

Un client per comandare le lampadine Philips Hue

```
String baseURL = "http://localhost:8000"; // Oppure IP del Bridge
String username = "newdeveloper"; // Oppure stringa ottenuta dal Bridge
String lightsURL = baseURL + "/api/" + username + "/lights/";

RestTemplate rest = new RestTemplate();
```

La classe RestTemplate implementa l'interfaccia RestOperations (specifica un insieme di operazioni RESTful)

https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html

Metodi per le operazioni GET e PUT

```
Map<String, ?> allLights = rest.getForObject(lightsURL, HashMap.class);
```

Il metodo getForObject(getForObject(URI url, Class<T> responseType) fa una GET sulla URL specificata, nel nostro caso ottiene la descrizione completa di tutte le lampadine.

Il server risponde in JSON: le informazioni in esso contenute vengono restituite dal metodo sotto forma di una mappa con:

chiave <id lampadina> e valore <descrizione lampadina>

La descrizione della lampadina a sua volta è articolata in una gerarchia.

Metodi per le operazioni GET e PUT

```
if (allLights != null) { // accendiamo tutte le lampadine impostando l'effetto colorloop
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        String colorloop = "{\"on\":true, \"effect\": \"colorloop\" }";
        HttpEntity<String> request = new HttpEntity<>(colorloop,headers);
```

La classe HttpHeaders rappresenta l'header di una richiesta/risposta http

Il metodo setContentType specifica come è rappresentato il body (**APPLICATION_JSON**)

La classe HttpEntity rappresenta una richiesta/risposta http e consiste di header e body. Al costruttore passiamo il body (stringa *colorloop*) e l'header (*headers*).

Nota: l'effetto colorloop si vede sulle lampadine vere, non sull'emulatore. Provate a costruire un body diverso per impostare il colore anziché l'effetto.

Metodi per le operazioni GET e PUT

```
for (String light : allLights.keySet()) { // per ogni lampadina nella mappa
                                          // (la chiave è l'id della lampadina)
        {   String callURL = lightsURL+light + "/state";
            rest.put(callURL,request);  }
    }
```

Il metodo put genera una richiesta di tipo PUT sullo stato della lampadina light con richiesta di impostare l'attributo "effect" a "colorloop"

Metodi per le operazioni GET e PUT

Inserire un ritardo per attendere qualche secondo prima di spegnere

```
String off = "{\"on\":false }";
HttpEntity<String> requestOff = new  HttpEntity<>(off,header)
 for (String l : allLights.keySet())
            { // modificare lo stato di tutte le lampadine "on": false
              String callURL = lightsURL+l + "/state";
               rest.put(callURL,requestOff);
            }
```

Create il progetto in IntelliJ, inserite le dipendenze, copiate il codice poi eseguite avendo prima attivato l'emulatore.

```
public class EsperimentiHue {
    public static void main(String[] args) {
        String baseURL = "http://localhost:8000";
        String username = "newdeveloper";
        String lightsURL = baseURL + "/api/" + username + "/lights/";

        RestTemplate rest = new RestTemplate();

        Map<String, ?> allLights = rest.getForObject(lightsURL, HashMap.class);

        if (allLights != null) {
            HttpHeaders headers = new HttpHeaders();
            headers.setContentType(MediaType.APPLICATION_JSON);
            String colorloop = "{\"on\":true, \"effect\": \"colorloop\" }";
            HttpEntity<String> request = new HttpEntity<>(colorloop,headers);
            for (String light : allLights.keySet())
              {   String callURL = lightsURL+light + "/state";
                    rest.put(callURL,request);
              }

            for (int i=0;i<10;i++)
              { // ATTENDE 10 SECONDI PRIMA DI SPEGNERE
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) { e.printStackTrace(); }
                    System.out.println(10-i);
              }
             String off = "{\"on\":false }";
             HttpEntity<String> requestOff = new HttpEntity<>(off,headers);
             for (String l : allLights.keySet()) {
                        String callURL = lightsURL+l + "/state";
                        rest.put(callURL,requestOff);
                }
            }
        }
}
```

Esempio creato da Luigi De Russis

ESERCIZIO

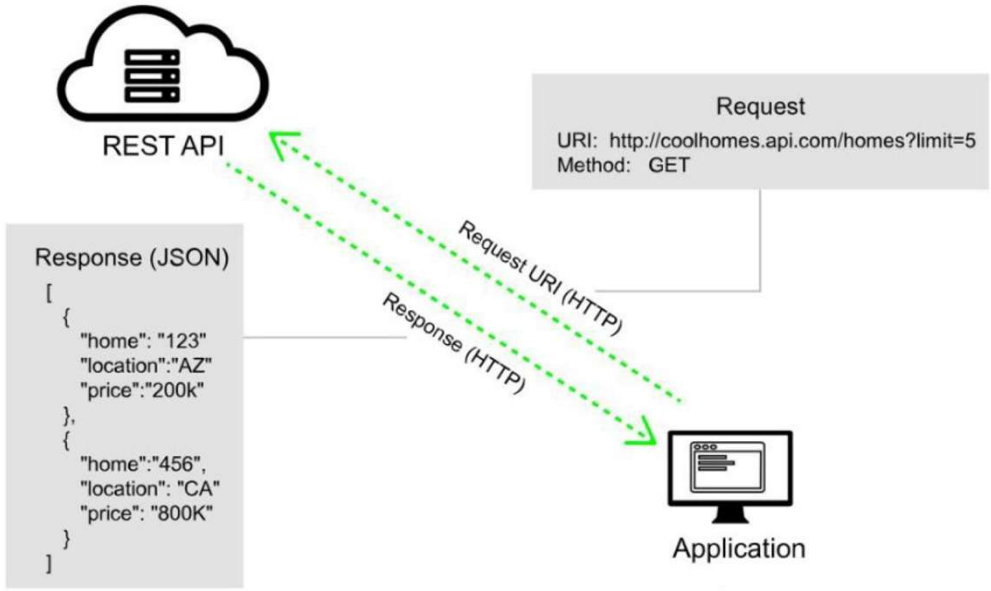
- Modificate l'esempio in modo che quando viene accesa ciascuna lampadina il suo colore gradualmente si modifichi (impostando l'attributo hue ad un valore che di volta in volta viene incrementato di 1000); dopo la prima fase le lampadine dovranno avere colori diversi
- Alla fine, quando le lampadine devono essere spente, inserire un ciclo decrementando l'attributo bri (inizialmente impostato al valore massimo) ad ogni iterazione e facendo il put del nuovo valore.

REpresentational State Transfer

Stile architetturale per applicazioni distribuite.

Interazione di tipo client-server veicolato tramite HTTP

Il server espone delle risorse identificate tramite URI; client e server si scambiano una rappresentazione della risorsa



Le operazioni CRUD si realizzano utilizzando i metodi POST, GET, PUT e DELETE di HTTP