Politecnico di Torino

Relazione del primo laboratorio software del corso “Tecnologie per IoT”

Pietro Macori s246163

Alessandro Versace s246056

Ferdinando Micco s245340

ANNO ACCADEMICO 2019-2020

Sommario

Il laboratorio software del corso “Tecnologie per IoT” è suddiviso in quattro esercizi, basati sull’utilizzo del linguaggio Python, la programmazione ad oggetti e sul framework “cherrypy” per la realizzazione di REST web services. Lo scopo generale consiste nel prendere confidenza con vari metodi REST implementabili per la conversione di temperature tra unità di misura differenti, l’utilizzo di un formato JSON per lo scambio di dati relativi alla temperatura e per salvare lo stato di una dashboard. Il report è suddiviso in quattro parti, ognuna relativa al corrispettivo esercizio, nei quali verrà analizzata l’implementazione del codice.

Esercizio 1

Nel primo esercizio è richiesta l’implementazione di web services in RESTful-style per la conversione di un singolo valore di temperatura da un’unità di misura iniziale a una finale. Questi tre dati devono essere fornibili tramite una richiesta HTTP GET sotto forma di parametri (query) al webserver.

Le unità di misura compatibili sono Celsius ‘C’, Kelvin ‘K’ e Fahrenheit ‘F’, e l’output verrà fornito sotto forma di report JSON validabile, quindi compatibile con la sintassi di questo formato. Inoltre è richiesta una gestione dei principali errori riscontrabili.

Esempio di richiesta: http://localhost:8080/**converter**?**value**=10&**originalUnit**=C&**targetUnit**=K

```python
import cherrypy
import json

class Ex1Site(object):
    exposed=True

    def GET(self,*uri,**params):
        if len(params)!=3:
            raise cherrypy.HTTPError(400,"The uri is not satisfied [/value/originalUnit/targetUnit]")


        originalUnit=params["originalUnit"]
        targetUnit=params["targetUnit"]

        try:
            value=float(params["value"])
            errorConversion=0
        except (ValueError, TypeError):
            errorConversion=1

        if errorConversion==1:
            raise cherrypy.HTTPError(400,"The temperature must be an integer or a float")

        result=convertValue(originalUnit,targetUnit,value)


        d={}

        d["Original value"]=value
        d["Original unit"]=originalUnit
        d["Converted value"]=round(result,2)
        d["Converted unit"]=targetUnit

        return(json.dumps(d,indent=4))

```

In questa prima parte del codice mostriamo l’implementazione della classe *Ex1Site()* che sarà poi esposta al web e la relativa implementazione del metodo GET. Il funzionamento consiste

nell'estrapolazione dei dati utili dal dizionario fornito tramite ***params* con la richiesta HTTP GET, nelle variabili originalUnit, targetUnit e value. Questi dati vengono poi passati alla funzione *convertValue()* che si occupa della conversione nel valore finale secondo le unità di misura fornite. Infine viene creato il dizionario di output, in cui si inseriscono i dati richiesti con arrotondamento alla seconda cifra dopo la virgola del valore convertito, che sarà poi ritornato sotto forma di stringa attraverso il metodo *dumps()* della libreria *json*. Il resto del codice tratta la gestione delle principali eccezioni sul formato dei parametri e sulla loro validità, con conseguente lancio di errori (400 e 404) relativi alla richiesta del client.

Implementazione della funzione di conversione:

```
38  def convertValue(originalUnit,targetUnit,value):
39      if originalUnit=='K':
40          if targetUnit=='C':
41              result=value-273.15
42          elif targetUnit=='F':
43              result=(value-273.15)*9/5+32
44          else:
45              raise cherrypy.HTTPError(404,"Target unit not found [C-K-F]")
46      elif originalUnit=='C':
47          if targetUnit=='K':
48              result=value+273.15
49          elif targetUnit=='F':
50              result=(value*9/5)+32
51          else:
52              raise cherrypy.HTTPError(404,"Target unit not found [C-K-F]")
53      elif originalUnit=='F':
54          if targetUnit=='K':
55              result=(value-32)*5/9+273.15
56          elif targetUnit=='C':
57              result=(value-32)*5/9
58          else:
59              raise cherrypy.HTTPError(404,"Target unit not found [C-K-F]")
60      else:
61          raise cherrypy.HTTPError(404,"Original unit not found [C-K-F]")
62
63      return result
```

Main con configurazione, mount della classe esposta e start dell'engine di *cherrypy*:

```
66  if __name__=="__main__":
67      conf = {
68      '/':{
69          'request.dispatch':cherrypy.dispatch.MethodDispatcher(),
70          'tools.sessions.on':True
71          }
72      }
73      cherrypy.tree.mount(Ex1Site(),'/converter',conf)
74
75      cherrypy.engine.start()
76      cherrypy.engine.block()
```

Esercizio 2

Questo esercizio ha le stesse finalità del precedente, con la differenza di uso di una richiesta HTTP GET con passaggio di parametri separati da "/":

Esempio: localhost:8080/converter/10/C/K

Per questo motivo le differenze si concentrano nell'implementazione della classe esposta *Ex2Site()* e il suo metodo GET.

```python
import cherrypy
import json

class Ex2Site(object):
    exposed=True
    def GET(self,*uri,**params):
        if len(uri)!=3:
            raise cherrypy.HTTPError(400,"The uri is not satisfied [/value/originalUnit/targetUnit]")

        try:
            value=float(uri[0])
            errorConvertion=0
        except (ValueError, TypeError):
            errorConvertion=1

        if errorConvertion==1:
            raise cherrypy.HTTPError(400,"The temperature must be an integer or a float")

        originalUnit=uri[1]
        targetUnit=uri[2]

        result=convertValue(originalUnit,targetUnit,value)

        d={}
        d["Original value"]=value
        d["Original unit"]=originalUnit
        d["Converted value"]=round(result,2)
        d["Converted unit"]=targetUnit

        return(json.dumps(d,indent=4))
```

I dati sono ricevuti nel parametro del metodo GET **uri* sotto forma di lista ordinata secondo l'ordine di invio dei parametri nella richiesta. Conoscendo a priori l'ordine di questi, si estrapolano come in precedenza i vari parametri essenziali per la conversione e la ricostruzione JSON dell'output, poi convertito in stringa.

Esercizio 3

Il terzo esercizio consiste invece in una variante dei precedenti dove la classe esposta vede l'implementazione di un metodo PUT che riceve un JSON nel body della richiesta. La richiesta ha gli stessi campi forniti in precedenza, con la differenza di una lista di valori *values* al posto di un singolo valore da convertire. Come in precedenza è richiesta una risposta da parte della classe in formato stringa compatibile JSON con la conversione dei valori nell'unità di misura *targetUnit*.

Esempio di contenuto di richiesta:

{
"values": [10, 9, 8, 7, 6, 5, 3, 2, 1],
"originalUnit ": "C",
"targetUnit ": "K"
}

```python
import cherrypy
import json

class Ex3Site(object):
    exposed=True

    def PUT(self,*uri,**params):
        d=json.loads(cherrypy.request.body.read())
        listValues=d["values"]
        originalUnit=d["originalUnit"]
        targetUnit=d["targetUnit"]


        listResults=convertValues(originalUnit,targetUnit,listValues)


        r={}

        r["values"]=listValues
        r["originalUnit"]=originalUnit
        r["convertedValues"]=listResults
        r["ConvertedUnit"]=targetUnit

        return(json.dumps(r,indent=4))

```

Il codice sopra mostra la classe esposta *Ex3Site()* con il corpo del metodo PUT. Il funzionamento consiste nel caricamento del JSON passato come richiesta HTTP PUT in un dizionario e il relativo salvataggio in variabili temporanee dei valori interni alle chiavi di interesse. La conversione dei valori è affidata alla funzione *convertValues(),* che è una variante della traduzione di valori di una intera lista che prende come parametri le unità di misura e la lista di lavori, ritornando la lista convertita.
Infine si costruisce un dizionario finale *r*, che poi tramite un *json.dumps()* viene convertito in stringa per l'output.
Alleghiamo la funzione di conversione evitando di includere il *main*, che non ha praticamente subito variazioni rispetto agli esercizi precedenti:

```python
def convertValues(originalUnit,targetUnit,listValues):
    listResults=[]
    for i in range(len(listValues)):
        try:
            value=float(listValues[i])
            errorConvertion=0
        except (ValueError, TypeError):
            errorConvertion=1

        if errorConvertion==1:
            raise cherrypy.HTTPError(400,"The temperature must be an integer or a float")

        if originalUnit=='K':
            if targetUnit=='C':
                result=value-273.15
                listResults.append(result)
            elif targetUnit=='F':
                result=(value-273.15)*9/5+32
                listResults.append(result)
            else:
                raise cherrypy.HTTPError(400,"Invalid target unit [C-K-F]")

        elif originalUnit=='C':
            if targetUnit=='K':
                result=value+273.15
                listResults.append(result)
            elif targetUnit=='F':
                result=(value*9/5)+32
                listResults.append(result)
            else:
                raise cherrypy.HTTPError(400,"Invalid target unit [C-K-F]")

        elif originalUnit=='F':
            if targetUnit=='K':
                result=(value-32)*5/9+273.15
                listResults.append(result)
            elif targetUnit=='C':
                result=(value-32)*5/9
                listResults.append(result)
            else:
                raise cherrypy.HTTPError(400,"Invalid target unit [C-K-F]")

        else:
            raise cherrypy.HTTPError(400,"Invalid original unit [C-K-F]")

        return listResults
```

Esercizio 4

L'ultimo esercizio di questo laboratorio richiede invece di interfacciarsi con *freeboard*, che mette a disposizione una serie di widget impostabili tramite una interfaccia grafica già fornitaci, e farne un deploy come web service usando *cherrypy*. Oltre a esporlo sul web tramite il metodo GET a seguito di una richiesta HTTP GET, è richiesto lo sviluppo di un metodo POST che, a seguito di una richiesta HTTP POST triggerabile dalla pagina *index.html*, permetta di salvare su un file *dashboard.json* i widget inseriti e la loro disposizione nel sito statico in formato JSON. L'utilità sta nella possibilità di poter effettuare un load di una qualsiasi dashboard precedentemente creata e poter continuare il lavoro o il monitoraggio precedente.

```python
if __name__=='__main__':

    conf = {
        '/':{
            'request.dispatch' : cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on' : True,
            'tools.staticdir.root': os.path.abspath(os.getcwd())
        },
        '/css':{
            'tools.staticdir.on': True,
            'tools.staticdir.dir': './freeboard/css'
        },
        '/js':{
            'tools.staticdir.on': True,
            'tools.staticdir.dir': './freeboard/js'
        },
        '/img':{
            'tools.staticdir.on': True,
            'tools.staticdir.dir': './freeboard/img'
        },
        '/plugins':{
            'tools.staticdir.on': True,
            'tools.staticdir.dir': './freeboard/plugins'
        }
    }

    cherrypy.tree.mount(Freeboard(), '/', conf)
    cherrypy.engine.start()
    cherrypy.engine.block()
```

La classe montata da esporre si chiama *Freeboard()* e la differenza principale con i precedenti esercizi sta nell'inserimento nel dizionario di configurazione *conf* dei vari percorsi in cui *cherrypy* può trovare le risorse richieste per il corretto funzionamento di freeboard (css, js, immagini e plugins vari).

La classe esposta invece si occupa semplicemente di ritornare l'*index.html* di *freeboard* dopo averlo aperto per il metodo GET.

Il metodo POST invece riceve di default il path del file di salvataggio della *dashboard.json*, che verrà aperto; successivamente verranno scritti sopra i valori contenuti nel dizionario ricevuto tramite richiesta HTTP POST e infine il file json verrà chiuso.

```python
import cherrypy
import os

class Freeboard():

    exposed = True
    def GET(self, *uri):
        return open('freeboard/index.html')

    def POST(self, *uri, **params):
        file = 'freeboard/dashboard/dashboard.json'
        f = open(file, "w")
        f.write(params['json_string'])
        f.close()
        return

```