Laboratorio TECNOLOGIE PER IOT – SW PARTE 1

Gatti Gabriele s245636

Iobbi Amedeo s228856

Esercizio 1

```
import cherrypy
import json

class TemperatureClient():
    exposed = True

    def fToC(self,t):
        if t < -459.67:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return (t-32) * 5.0/9.0

    def fToK(self,t):
        if t < -459.67:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return (t-32) * 5.0/9.0 + 273.15

    def cToF(self,t):
        if t < -273.15:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return (t * 9.0/5.0) + 32

    def cToK(self,t):
        if t < -273.15:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return t + 273.15

    def kToC(self,t):
        if t < 0:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return t - 273.15

    def kToF(self,t):
        if t < 0:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return (t - 273.15) * 9.0/5.0 + 32

    def GET(self, **params):

        tmpDict = {}
        try:
            if(len(params)!=3):
                raise SyntaxError("Wrong number of parameter")

            for key in params.keys():
                if not key in ("value", "originalUnit", "targetUnit"):
                    raise SyntaxError("Una delle chiavi e' scritta in modo sbagliato")

            tmpDict['value'] = int(params['value'])
            tmpDict['originalUnit'] = params['originalUnit']
            tmpDict['targetUnit'] = params['targetUnit']
```

```
            if tmpDict['originalUnit'] == 'F' and tmpDict['targetUnit'] == 'C':
                result = self.fToC(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'F' and tmpDict['targetUnit'] == 'K':
                result = self.fToK(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'C' and tmpDict['targetUnit'] == 'F':
                result = self.cToF(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'C' and tmpDict['targetUnit'] == 'K':
                result = self.cToK(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'K' and tmpDict['targetUnit'] == 'C':
                result = self.kToC(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'K' and tmpDict['targetUnit'] == 'F':
                result = self.kToF(tmpDict['value'])
            elif tmpDict['originalUnit'] == tmpDict['targetUnit']:
                result = tmpDict['value']
            else:
                raise SyntaxError("Unita' sbagliata")

            tmpDict['result'] = result

        except SyntaxError:
            raise cherrypy.HTTPError(400, "I valori da passare sono tre: value,
targetUnit, originalUnit")

        return json.dumps(tmpDict)

if __name__ == "__main__":

    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True
        }
    }

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

Nel primo esercizio bisogna progettare un servizio web che converta dei valori di temperatura in differenti unità di misura.
Per farlo utilizziamo cherrypy ed implementiamo un metodo GET che riceve come parametri tre valori: valore da convertire, unità di misura iniziale e unità di misura finale.

Tutte le operazioni di acquisizione e conversione dati vengono eseguite nella classe ***TemperatureClient()***. Nella funzione ***GET*** utilizziamo *****params*** perchè l'URL è nel formato http://localhost:8080/converter?value=10&originalUnit=C&targetUnit=K.

Racchiudiamo tutte le operazioni dentro un blocco ***try/catch*** per gestire eventuali eccezioni, come ad esempio se il numero di parametri passato non è corretto, se i nomi delle chiavi dei parametri non è uguale a quello scelto.

Infine nelle funzioni ***xToX()*** vengono eseguite le operazioni di conversione controllando che i valori di temperatura passati siano maggiori del valore minimo stabilito per quel tipo di scala.

Alla fine utilizziamo ***json.dumps()*** per generare dal dizionario temporaneo creato l'output in stile json.

Le funzionalità sono state testate con l'estensione ***POSTMAN*** di Chrome e il codice json generato è stato testato e validato con ***jsonlint***

Esercizio 2

```
import cherrypy
import json

class TemperatureClient():
    exposed = True

    def fToC(self,t):
        if t < -459.67:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return (t-32) * 5.0/9.0

    def fToK(self,t):
        if t < -459.67:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return (t-32) * 5.0/9.0 + 273.15

    def cToF(self,t):
        if t < -273.15:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return (t * 9.0/5.0) + 32

    def cToK(self,t):
        if t < -273.15:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return t + 273.15

    def kToC(self,t):
        if t < 0:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return t - 273.15

    def kToF(self,t):
        if t < 0:
            raise ValueError("Valore minimo di temperatura superato")
        else:
            return (t - 273.15) * 9.0/5.0 + 32

    def GET(self, *uri):

        tmpDict = {}
        try:
            if(len(uri)!=3):
                raise SyntaxError("Wrong number of parameter")

            tmpDict['value'] = int(uri[0])
            tmpDict['originalUnit'] = uri[1]
            tmpDict['targetUnit'] = uri[2]

            if tmpDict['originalUnit'] == 'F' and tmpDict['targetUnit'] == 'C':
                result = self.fToC(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'F' and tmpDict['targetUnit'] == 'K':
                result = self.fToK(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'C' and tmpDict['targetUnit'] == 'F':
                result = self.cToF(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'C' and tmpDict['targetUnit'] == 'K':
                result = self.cToK(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'K' and tmpDict['targetUnit'] == 'C':
                result = self.kToC(tmpDict['value'])
            elif tmpDict['originalUnit'] == 'K' and tmpDict['targetUnit'] == 'F':
```

```
                result = self.kToF(tmpDict['value'])
            elif tmpDict['originalUnit'] == tmpDict['targetUnit']:
                result = tmpDict['value']
            else:
                raise SyntaxError("Unita' sbagliata")

            tmpDict['result'] = result

        except SyntaxError:
            raise cherrypy.HTTPError(400, "I valori da passare sono tre: value,
targetUnit, originalUnit")

        return json.dumps(tmpDict)

if __name__ == "__main__":

    conf = {
        '/': {
            'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
            'tools.sessions.on': True
        }
    }

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

L'esercizio due è una modifica del primo esercizio, ovvero l' URL passato non è più nella forma: http://localhost:8080/converter?value=10&originalUnit=C&targetUnit=K ma è nella forma: http://localhost:8080/converter/10/C/K.

Quindi per poter acquisire i valori non possiamo più utilizzare ****params***, ma utilizziamo ****uri***. ****uri*** può essere gestito come un array, quindi controlliamo sempre che i valori siamo tre, altrimenti lanciamo un'eccezione e successivamente salviamo come nel caso precedente tutti i valori in un dizionario temporaneo, sapendo che nella posizione 0 dell'array c'è il valore di temperatura da convertire, nella posizione 1 l' unità di misura di partenza e nella posizione 2 l'unità di misura finale.

Infine nelle funzioni ***xToX()*** vengono eseguite le operazioni di conversione controllando che i valori di temperatura passati siano maggiori del valore minimo stabilito per quel tipo di scala.

Alla fine utiliziamo json.dumps() per generare dal dizionario temporaneo creato l'output in stile json.

Le funzionalità sono state testate con l'estensione POSTMAN di Chrome e il codice json generato è stato testato e validato con jsonlint

Esercizio 3

```
import cherrypy

class TemperatureClient():
    exposed = True

    def celsius2kelvin(self, t):
        if t < -273.15:
            raise Exception
        return t + 273.15

    def kelvin2celsius(self, t):
        if t < 0:
            raise Exception
        return t - 273.15

    @cherrypy.tools.json_in()
    @cherrypy.tools.json_out()
    def PUT(self, *uri, **params):
        try:
            json_temps = cherrypy.request.json
            print(json_temps)
            values = json_temps["values"]
            originalUnit = json_temps["originalUnit"]
            targetUnit = json_temps["targetUnit"]

            if originalUnit == "C" and targetUnit == "K":
                targetFunc = self.celsius2kelvin
            elif originalUnit == "K" and targetUnit == "C":
                targetFunc = self.kelvin2celsius
            else:
                raise Exception

            targetValues = []
            for v in values:
                targetValues.append(targetFunc(v))
            result = json_temps
            result["targetValues"] = targetValues

            return result

        except:
            errorMsg = "Your request: {} ".format(json_temps)
            errorMsg += "Correct example: {'originalUnit': 'C', 'targetUnit': 'K', 'values
': [1,2,3,4]} "
            errorMsg += "C or K are only units available."

            raise cherrypy.HTTPError(400, errorMsg)

if __name__ == "__main__":
    conf={
        '/':{
```

```
            'request.dispatch':cherrypy.dispatch.MethodDispatcher(),
            'request.show_tracebacks': False,
        }
    }
    cherrypy.tree.mount(TemperatureClient(),'/temperaturePUT',conf)
    cherrypy.engine.start()
    cherrypy.engine.block()
```

L'obiettivo dell'esercizio è convertire una lista di valori di temperatura da gradi Celsius a gradi Kelvin e vice versa, tale lista è ottenuta mediante metodo PUT che fornisce dati in formato JSON allo script di backend, esso si occuperà quindi dell'elaborazione degli stessi effettuando i dovuti controlli.

La classe `TemperatureClient` viene esposta sul path `/temperaturePUT`, l'unico gestore definito è quello per il metodo PUT che grazie al decoratore `@cherrypy.tools.json_in` ottiene la componente JSON della richiesta, verifica le unità di misura e la loro correttezza e a seconda delle suddette richiama gli helper methods `celsius2kelvin` o `kelvin2celsius` che eseguono le conversioni, controllando l'esattezza dei valori ricevuti. I dati ricevuti vengono poi riuniti in un dizionario con i dati elaborati e restituiti in formato JSON grazie al secondo decoratore `@cherrypy.tools.json_out`.

Le eccezioni scatenate all'interno del metodo PUT possono essere determinate esclusivamente da errori nel formato della richiesta, pertanto in tal caso viene istanziata la pagina di errore "400: Bad Request", con un messaggio di errore personalizzato che fornisce all'utente informazioni su come strutturare la richiesta. La visualizzazione del TraceBack dell'eccezione viene disabilitata nella configurazione per ragioni di sicurezza.

Le funzionalità sono state testate con l'estensione RESTED di Firefox (https://addons.mozilla.org/it/firefox/addon/rested/), inserendo l'header 'Content-Type':'application/json'.

Esercizio 4

```
import cherrypy
import os
import json
from datetime import datetime

class FreeboardDeployer():
    exposed = True

    def GET(self, *uri, **params):
        return open("freeboard/index.html")

    def POST(self, *uri, **params):
        if uri == ('saveDashboard',):
            config = cherrypy.request.params["json_string"]
            time = datetime.now()
            timestamp = str(time.year) + str(time.month).rjust(2, "0") + str(time.day).rju
st(2, "0")
            timestamp += str(time.hour).rjust(2, "0") + str(time.minute).rjust(2, "0") + s
tr(time.second).rjust(2, "0")
            with open("./freeboard/dashboard/Dashboard_"+timestamp+".json", "w") as out:
                out.write(config)
        else:
```

```
            raise cherrypy.HTTPError(404)


if __name__ == "__main__":
    conf={
        '/':{
            'request.dispatch':cherrypy.dispatch.MethodDispatcher(),
            'tools.staticdir.on': True,
            'tools.staticdir.root': os.path.abspath(os.getcwd()),
            'tools.staticdir.dir': 'freeboard/',
            'request.show_tracebacks': False,
        }
    }
    cherrypy.tree.mount(FreeboardDeployer(),'/',conf)
    cherrypy.engine.start()
    cherrypy.engine.block()
```

L'obiettivo dell'esercizio è la gestione di contenuti statici mediante cherrypy, inoltre è necessario occuparsi del salvataggio dei contenuti creati mediante questi contenuti, è necessario quindi istanziare la piattaforma Freeboard e gestire le richieste da essa provenienti.

Mediante la configurazione la cartella `freeboard/` viene esposta, inoltre ogni richiesta GET effettuata su qualsiasi path del server determinerà l'istanziazione della pagina `freeboard/index.html` da cui è possibile iniziare la creazione della propria dashboard.

Al contrario le richieste POST vengono consentite in un unico path, corrispondente a `/saveDashboard`, qualsiasi altra POST in un path differente determinerà la visualizzazione della pagina di errore "404: Not Found". Questa locazione viene interrogata nel momento in cui dalla pagina `freeboard/index.html` viene effettuata la richiesta del salvataggio della dashboard appena configurata. Questa richiesta contiene un unico parametro: `json_string`, a cui viene attribuito come valore la stringa JSON contenente la configurazione della dashboard che si desidera salvare. A questo punto, allo scopo di non sovrascrivere eventuali file creati in precedenza viene generato un timestamp, e successivamente viene aperto il file `/freeboard/dashboard/Dashboard_timestamp.json`, in modalità di scrittura. La stringa JSON viene quindi scritta sul file, salvando la configurazione come richiesto.