TP #2 d'introduction au Cloud Computing M2CCI

Université Claude Bernard – Lyon 1 / Département Informatique

INTRODUCTION AUX ARCHITECTURES WEB ORIENTÉES SERVICES

Objectifs : L'objectif de ce TP est de proposer une introduction aux architectures orientés Services (*Service Oriented Architecture* (SOA)), très populaire pour les services web et cloud, et à l'architecture Model Vue Contrôleur (MVC), très populaire dans l'industrie. Pour cela nous vous proposons de développer deux services web, l'un donnant l'heure et la date du jour, et l'autre, récupérant cette date et cette heure pour en générer une page web.

Ces services web seront développés en **Python** un langage de programmation orienté objet. Ce TP servira donc d'introduction à Python un langage particulièrement populaire. Il est conçu prioritairement pour la lisibilité du code.

Définitions :

MVC : MVC est une architecture d'application très populaire pour le développement d'application web et local exploitant le paradigme orienté objet. L'idée est de séparer le modèle (représentation de l'information) et la partie vue (interface graphique) par le contrôleur. L'intérêt est de développer séparément la partie applicative de la partie visuelle.

SOA (*service oriented architecture*) : l'architecture orienté service est un autre paradigme de programmation qui distribue sur plusieurs machines les opérations (services) de l'application. Les communications se font via des requêtes http.

REST (*REpresentational State Transfer*) : les services REST sont des services communiquant sur le web utilisant de simples requêtes http utilisant les « verbes » http (`POST, GET, PUT, DELETE`). Pour ce TP nous n'utiliserons que le verbe GET.

PYTHON VS JAVA

	java	Python
Indentation	{ …}	: [Tab] …
Class	class NomClass	class NomClass :
Constructeur	NomClass(type var)	__init__(self, var) :
Héritage	Extends	Class ClassFille(ClassMere):
Interface	Implement	Absent utilise l'héritage multiple
Function	Type nomMethode(type var)	def nomMethode(var)
Variable	Type nomVariable	nomVariable
Curent instance	this	self (obligatoire)
Class mère	super	ClassMere
Import	import package.class	import class from package
Commentaire ligne simple	//	#
Commentaire ligne multiple	/*….*/	"""…."""
String ligne multiple		nomString = """…."""

PARTIE 1 : SERVICE *TIME* ET *DATE*

Démarrer sous GNU/Linux (Debian). Créez le dossier `time (mkdir time)`

Modèle

Dans le dossier `time` créez un fichier avec l'éditeur de texte de votre choix **`model.py`**

Dans ce fichier vous allez créer une classe avec les méthodes retournant les variables de l'heure et de la date. Pour cela vous utiliserez la méthode `time.strftime()`

http://www.tutorialspoint.com/python/time_strftime.htm

Créez 6 méthodes retournant respectivement l'heure, les minutes, les secondes, le jour, le mois et l'année.

```
#! /usr/bin/env python3
import time

class Model:
    def getMinute(self) :
            return time.strftime("%M")    # retourne minute courante
    def getSecond(self) :
            return  time.strftime("...")  # retourne seconde courante
    def getHour(…
            …                             # retourne l'heure courante

# Une section main pour valider le bon fonctionnement
if __name__ == "__main__":
    print(Model().getMinute())
    print(Model().getSecond())
    ...
```

Contrôleur de Vues

Web.py est une bibliothèque simple pour les applications web en Python (http://webpy.org/). À l'aide des commandes suivante installez la bibliothèque `web.py`

```
sudo apt update
sudo apt install python3-pip
sudo pip3 install web.py
```

Toujours dans le dossier `time`, créez un fichier **`controler.py`**.

Dans ce fichier importer `web`, `json` et les classes de votre modèle

```
import web
import json
from model import Model
```

Web.py fonctionne en créant un *mapping* entre les urls et les classes associés. Pour cela on utilise une variable généralement nommée « `urls` ». Exemple : (attention au double quote si vous faite un copier/coller).

```
urls = ( "/hello", "Hello", "/date", "Date", "/time", "Time" )
```

Lors d'une requête http sur `http://machine:8080/hello`, web.py renvoie la valeur de retour de la méthode du même nom du verbe utilisé.

Note : par défaut un navigateur utilise `GET`

```
class Hello:
    def GET(self) :
          return "Hello world !"
```

Terminez votre fichier avec :

```
app = web.application (urls, globals())
if __name__ == "__main__":
   app.run()
```

Vous pouvez tester avec la commande python `controler.py` et en paramètre le port 8080
Exemple : `python3 controler.py 8080`
Dans votre navigateur : `http://192.168.239.xxx:8080/hello`
celas devrait afficher « Hello world ! » */!\ NE PAS METTRE DE SLASH A LA FIN de hello* !

JSON est une bibliothèque python permettant de *parser* un document au format *json*.
`json.dumps(dico)` prend en entré un *dictionnaire* en python et retourne une *string* formatée en json.

Les dictionnaires en python sont comme les tables sauf que l'on nomme l'index (exemple : `dico["key"]="value"`). Ils sont déclarés avec des accolades {}. Exemple : `{"key" :"value"}`

Réalisez un serveur Web retournant au format JSON, l'heure ou la date selon l'url saisie.
Exemples : `http://machine:8080/date` renvoit
`{"year": "2019", "month": "1", "day": "31"}`

`http://machine:8080/time` renvoit
`{"hour": "10", "minute": "40", "second": "30"}`

PARTIE 2 : SERVICE HTML

Créez un dossier `html (mkdir html)`

Modèle

Créez un fichier `model.py`

Importez-y la bibliothèque `requests` (une bibliothèque permettant de créer des requêtes http en langage python) que vous aurez préalablement installé avec : `sudo pip3 install requests`

```
import requests
```

Utilisez `requests.get(url)` pour récupérer les données de votre service « time » et « date » (cf. partie 1).
Utilisez `json.loads(json)` pour reconvertir en dictionnaire python.

Créez une classe avec deux méthodes : L'une récupérant l'heure, l'autre récupérant la date.
Exemple :

```
def getTime(self):
        data = requests.get("http://adresseIP:8080/time")
        return json.loads(data.text)
```

Contrôleur de Vues

Créez un fichier `controler.py`
Importez-y `web` et les classes de votre *model*.
Créez deux pages web l'une donnant l'heure et l'autre donnant la date.
Terminez également votre fichier avec :

```
app = web.application (urls, globals())
   if __name__ == "__main__":
        app.run()
```

Vous pouvez tester avec la commande python `controler.py` avec en paramètre le port 8443 différent de celui choisi dans la partie 1 si vous exécutez les services sur la même machine.

Exemple : `./controler.py 8443`

Dans un navigateur `http://machine:8443/time` devra afficher la date (ou l'heure) au format HTML.

Démo : http://192.168.239.xxx:8443/time

PARTIE 3 : VARIABLE REST (FACULTATIF)

Vous pouvez ajouter des variables à l'url de vos méthodes GET comme ceci :

```
http://mamachine:port/url1?var1=hello&var2=world
```

`web.input()` retourne un objet avec toutes les variables de la requête. Ceci fonctionne pour les méthodes GET et POST.

```
i= web.input()
print i.var1 # affiche hello
print i.var2 # affiche world
```

Exemple simple :

```
import web

urls = ('/hello', 'hello')
app = web.application(urls, globals())

# python test.py 8080
# http://localhost:8080/hello?name=titi
class hello:
    def GET(self):
            i=web.input()
            return 'Hello, ' + i.name + ' !' # Affiche Hello, titi !

if __name__ == "__main__":
    app.run()
```

Ajout d'input à la page html

La balise `<form>` permet de générer de nouvelles requêtes depuis le navigateur

```
<form action="getname">
    <input type="text" name="name"/>
    <input type=”submit” name=”submit”/>
</form>
Lorsque l'on clique sur le bouton le navigateur générera la requête ci-dessous.
GET  http://machine:port/getname?name=texteSaisi&change=Submit
```