TP JEE (3)
WebServices REST

Table des matières

1 Les WebsServices REST

Un Web Service est une fonctionnalité exposée sur le web à d'autres applications. Les web services reposent sur des protocoles et formats standards afin de faciliter l'intégration avec d'autres logiciels écrits dans des langages de programmation différents. Ce TP est une introduction aux services web de type REST (REpresentational State Transfer), une alternative est SOAP.

Les web services REST (REpresentational State Transfer) se basent sur les technologies fondatrices du web comme HTTP, XML, les URLs, etc. Ainsi, chaque service web REST est défini par une URL et une opération HTTP. Pour rappel, il existe quatre opérations HTTP : GET(lecture), POST (pour la création), PUT (pour la modification) et DELETE (pour la suppression). Une requête sur un service web se fait donc sur une URL en précisant le type d'opération HTTP, le format des données envoyées dans la requête (XML, JSON, text), et bien entendu le contenu des données envoyées. Voici quelques exemples :

```
// Recupere le contact ayant pour identifiant 1
Requete :
GET /rest/contact/1
HEADERS  Accept:application/xml
```

```
Reponse :
HEADERS  Content-Type:application/xml
CONTENU
<contact>
        <id>1234</id>
        <name>Adam</name>
</contact>
```

```
// Ajoute un contact
Requete :
POST /rest/contact
HEADERS  Accept:application/xml
<contact>
        <name>Adam</name>
</contact>


Reponse :
HEADERS Content-Type:application/xml
CONTENU
<contact>
        <id>1234</id>
        <name>Adam</name>
</contact>
```

```
// Modifie un contact
Requete :
PUT /rest/contact
HEADERS  Accept:application/xml
<contact>
        <id>1234</id>
        <name>AdamModif</name>
</contact>

Reponse :
HEADERS Content-Type:application/xml
CONTENU
<contact>
        <id>1234</id>
        <name>AdamModif</name>
</contact>
```

---

2 Jax-RS

2.1 Helloworld

1. Créez une nouvelle classe appelée ContactWebRESTApp dans le package org.iut.contact, comme le montre l'extrait de code suivant :

```
@ApplicationPath("/rest")
public class ContactWebRESTApp extends Application {
}
```

   Cette classe permet de spécifier à JAXRS le préfix des URLs de l'ensemble des services web. Ici ce sera /rest.
2. Créez un nouveau package org.iut.contact.rest
3. Dans ce nouveau package, créez une nouvelle classe appelée ContactResource.
4. Annotez la classe avec @Path("/contact") qui permet de donner un préfix aux URL des services exposées par cette classe.
5. Ajoutez une nouvelle méthode hello() retournant la chaine de caractère hello.
6. Annotez la méthode par @GET (pour spécifier que la méthode répond à l'opération HTTP GET) et @Path("/hello") pour spécifier l'URL de la méthode. Cette méthode sera donc exécutée pour toute opération HTTP de type GET sur l'URL /rest/contact/hello.
7. Déployez.
8. Testez avec votre navigateur.
9. Testez avec la commande curl de la façon suivant :

```
curl http://localhost:8080/ContactWeb-1.0-SNAPSHOT/rest/
   contact/hello
```

2.2 La manipulation des contacts

Nous allons donc transférer des instances de la classe Contact sur le réseau sous forme de flux XML ou JSON. Pour convertir automatiquement un contact en XML ou JSON, nous utiliserons la librairie JAXB. Cette dernière propose plusieurs annotations. Celle qui nous intéresse dans ce TP est @XmlRootElement qui permet de spécifier à JAXRS que les instances de la classe annotée peuvent être converti (sérialisées, marshallées) par JAXB. Cette annotation se place donc sur une classe.

1. Modifiez la classe contact afin qu'elle soit interprétée par JAXB.

2.2.1 Récupérer un Contact

1. Ajoutez, dans la classe ContactResource, une méthode permettant de retourner un Contact par son identifiant (le prototype est Contact get (Integer id)). Cette classe doit bien entendu utiliser la classe ContactService.

2. Cette méthode doit être accessible par l'opération HTTP GET et L'URL /rest/contact/ID. Pour apparier un paramètre passé dans l'URL avec un paramètre de méthode (ici ID) deux annotations sont nécessaires :
   - la première, @Path à laquelle il est possible de préciser un paramètre d'URL : @Path("/nom_parametre"). Cette annotation se place sur la méthode.
   - la seconde, @PathParam("nom_parametre") permet d'associer le paramètre de l'URL vers un paramètre de la méthode :

```
public Personne get (@PathParam("nom_parametre") Integer
    param)
```

3. Déployez et testez.
4. Le contact n'est pas "transformé" en XML. Pour corriger cela, utilisez l'annotation @Produces(MediaType.APPLICATION_XML) qui se place soit sur une classe soit sur une méthode.
5. Déployez et testez.
6. L'annotation @Produces permet de spécifier une liste de formats, il est donc possible d'utiliser plusieurs formats pour structurer le contact. An plus de XML utilisez JSON.
7. La sélection du format se fera alors à la demande sur client en spécifiant dans les headers de la requête HTTP, le format attendu par le client. Ce header s'appelle "Accept". Voici comment le spécifier avec curl :

```
curl -H "Accept:application/json" http://localhost:8080/
    ContactWeb-1.0-SNAPSHOT/rest/contact/ID
// ou
curl -H "Accept:application/xml" http://localhost:8080/
    ContactWeb-1.0-SNAPSHOT/rest/contact/ID
```

8. Déployez et testez.

2.2.2 Les autres opérations

Pour ajouter un contact, nous allons utiliser la méthode HTTP POST (et donc l'annotation @POST). Nous allons donc envoyer au serveur un contact représenter dans un flux XML ou JSON. Ce flux sera automatiquement interprété par JAXRS et convertit dans une instance de la classe contact. Il est néanmoins nécessaire de préciser quel type de format est accepté par le serveur via l'annotation @Consumes :

```
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON
    })
```

Il sera aussi nécessaire de préciser dans la requête HTTP quel type de format est envoyé. Nous utiliserons le Header Content-Type. Voici un exemple avec curl :

```
curl -X POST  \
        -H 'Content-Type: application/json' \
        -H 'Accept: application/json'  \
```

```
-d '{"prenom":"nouvellepersonne"}' \
http://localhost:8080/PersonneApp/rest/personne/
```

1. Créez un service REST permettant d'ajouter un contact.
2. Déployez et testez.
3. Ajouter des services REST pour supprimer et modifier un contact. Dans le fichier de réponse, donnez les commandes curl à exécuter.

3 Consommation des web services REST avec Python

Nos services web exposent des méthodes permettant de manipuler des contacts via d'autres langages. L'objectif de cette section est de créer des scripts (console ou graphique) permettant d'utiliser ces services en Python. Pour cela nous allons utiliser trois librairies standard de python à savoir httplib, urllib et json. La première chose à faire dans vos scripts est donc :

```
import httplib, urllib, json
```

La librairie httplib permet d'exécuter des requêtes HTTP. Voici un exemple de GET :

```
headers = {"Content-type": "application/json", "Accept": "
   application/json"}

conn = httplib.HTTPConnection("monserveur:8080");

conn.request("GET", "/PersonneWeb/rest/personne/1", "", headers);
response = conn.getresponse()
print response.status, response.reason
data = response.read()
print data
```

La librairie json va ensuite nous permettre de convertir la réponse (au format JSON) en objet et d'afficher ses attributs :

```
personne = json.loads(data);
print "ID : " + str(personne["id"]);
print "Firstname : " + str(personne["prenom"]);
```

Pour ajouter un contact, il est nécessaire de le construire avec les bons attributs, de le convertir en python via la méthode json.dumps et pour finir de l'envoyer en POST :

```
contactToAdd = {'firstname':"Test"}
conn.request("POST", "/ContactWeb-1.0-SNAPSHOT/rest/contact",
    json.dumps(contactToAdd), headers);
```

Une dernière chose ! Pour demander à l'utilisateur de rentrer une valeur, utilisez la fonction raw_input("Message :") :

```
prenom = raw_input("Quel est votre prenom :")
personne = {'prenom':prenom}
```

Vous pouvez aussi utiliser des composants graphiques avec la librairie Tkinter.

Exercice :

Créez un script python capable d'ajouter, récupérer, modifier ou supprimer un ou plusieurs contacts selon la demande de l'utilisateur. Essayez d'encapsuler les appels HTTP et la conversion JSON dans une classe que l'on pourra appeler ContactService.