TP 4A - Java Entreprise Edition.
Service Web RESTFul avec JAX-RS

Nicolas Delanoue et Sylvain Joyeau

2021

Le but de ce TP est que vous vous familiarisez avec la spécification JAX-RS avec l'implémentation Jersey.

Exercice 1 (Utilisation de `curl`)

1. Parcourez la page Web :
   `https://geo.api.gouv.fr/decoupage-administratif/communes`
2. Avec votre navigateur, trouvez le nombre d'habitants vivant dans votre ville préférée.
3. Ouvrez une invite de commande.
4. Avec l'aide de la commande curl, retrouvez cette même information.

Exercice 2 (JAX-RS Hello World et Calcul Imc)
Dans cet exercice, nous allons créer une API REST qui s'exécutera sur un serveur Tomcat. Cette API ne générera que des documents texte.

1. Lancez Eclipse IDE for Enterprise Java Developers.
2. Créez une application Web Dynamique nommée `TP5Ex2`.
3. Téléchargez et décompressez l'implémentation de la spécification JAX-RS nommé jersey 2.33 via `https://eclipse-ee4j.github.io/jersey/`.
   Indications : Il s'agit d'un fichier nommé `jaxrs-ri-2.33.zip`.
4. Copiez tous les fichiers `jar` contenus dans l'archive `jaxrs-ri-2.33.zip` vers le dossier `WebContent/WEB-INF/lib/`.
5. Copiez les trois fichiers `jar` téléchargés depuis l'url `http://perso-laris.univ-angers.fr/~delanoue/polytech/jee/jaxb.zip` vers le dossier `WebContent/WEB-INF/lib/` de votre application Web.
6. Ajoutez tous ces fichiers `jar` au Build Path de votre projet.
7. Créez un package nommé `ImcPack`.
8. Créez un fichier `web.xml` dans le répertoire `WebContent/WEB-INF/` à partir du modèle suivant.

```
1  <web-app version="3.0"
2    xmlns="http://java.sun.com/xml/ns/javaee"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4
5  <servlet>
6   <servlet-name>Jersey Web app</servlet-name>
```

```
7   <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
8   <init-param>
9     <param-name>jersey.config.server.provider.packages</param-name>
10    <param-value>_________</param-value>
11  </init-param>
12  <init-param>
13    <param-name>jersey.config.server.provider.scanning.recursive</param-name>
14    <param-value>true</param-value>
15   </init-param>
16  <load-on-startup>1</load-on-startup>
17 </servlet>
18
19 <servlet-mapping>
20   <servlet-name>Jersey Web app</servlet-name>
21   <url-pattern>/______/*</url-pattern>
22 </servlet-mapping>
23
24 </web-app>
```

9. A la ligne 10, précisez, avec la bonne casse, le package qui proposera le code traitant les requêtes REST.
10. A la ligne 21, précisez le préfixe `api` de l'URI qui permettra de solliciter notre API Rest.
11. Créez une classe `imcService` dans le package `imcPack`.
12. Ajoutez à cette classe une méthode `String ServiceHelloWorld()` qui retourne la chaîne de caractères `"hello World"`.
13. Annotez la classe `imcService` avec une annotation `@Path("/imc")`.
14. Annotez la méthode `public String ServiceHelloWorld()` avec les annotations suivantes : `@Path("/hello")`, `@GET` et `@Produces(MediaType.TEXT_PLAIN)`.
15. Effectuez avec votre navigateur une requête `GET` du protocole `http` vers l'url `http://localhost:8080/TP5Ex2/api/imc/hello` afin de tester le bon fonctionnement de votre API REST.
    Indication : dans la suite de ce TP, ce genre de requête sera noté au sens du protocole `http` sous la forme suivante :

    ```
    GET /TP5Ex2/api/imc/hello HTTP/1.1
    ```

16. Ajoutez à votre package la classe `imc.java` avec le contenu suivant :

    ```
    package imcPack;
    public class imc {
      private double poids ;
      private double taille ;
      public imc(double p, double t)
        { poids = p;
          taille = t;
        }
      public double calcul ()
        {
    ```

```
    return (poids/(taille*taille));
    }
}
```

17. Ajoutez une ressource à votre service de sorte que la requête http de la forme suivante :

    ```
    GET /TP5Ex2/api/imc/calculimc/90/1.86 HTTP/1.1
    ```

    produise, en réponse, le texte suivant :

    ```
    poids : 90
    taille :1.86
    imc :26.014568158168572
    ```

18. Testez avec `curl`.

Exercice 3 (Une API Rest qui fournit des données)
Dans cet exercice, nous allons mettre en place un service web REST qui fournira des données.

1. Créez une nouvelle application dynamique nommée `TP5Ex3`.
2. Reprenez les questions de l'exercice précédent pour insérer les librairies nécessaires à l'exploitation de jersey.
3. Créez un package nommé `appEtudiant`.
4. Dans ce package, créez une classe nommée `Etudiant` avec pour attributs :
   — `nom`,
   — `prenom`,
   — `groupe`,
   — `numeroEtudiant`,
   — `anneeDeNaissance`.
5. Faite en sorte que cette classe implémente l'interface `Serializable`.
6. Utilisez astucieusement Eclipse pour générer les getter et setter.
7. Créez une classe nommée `EtudiantService` dont les méthodes exécuteront les appels REST.
8. Ajoutez une méthode `testService` à cette classe avec pour code :

   ```
   public Etudiant testService()
   {
     Etudiant nico = new Etudiant("nico", "de", 1980, 007, "enseignants");
     return nico;
   }
   ```

9. Annotez correctement la classe `EtudiantService` et sa méthode `testService` de sorte qu'on obtienne un document JSON en sollicitant le web service REST via l'URI :

   ```
   GET /TP5Ex3/api/etudiant/test HTTP/1.1
   ```

10. Testez avec votre navigateur et avec la commande `curl`.
11. Ajoutez une classe nommée `EtudiantsRepertoire` au package `appEtudiant`.
12. Ajoutez l'attribut suivante à cette dernière classe

    ```
    private static Map<Integer, Etudiant> datas = null;
    ```

Cette classe et son attribut `datas` vont nous permettre de gérer une base de données d'étudiants "en mémoire". Le numéro étudiant de la classe `Etudiant` servira de clé à la map `data`.

13. Ajoutez un constructeur à la classe `EtudiantsRepertoire` qui initialise l'objet data avec des données :

```
datas = new HashMap<Integer, Etudiant>();
datas.put(0, new Etudiant("Delanoue", "Nicolas",1980, 0, "SAGI1"));
datas.put(1, new Etudiant("Joyeau", "Sylvain",1980, 1, "SAGI3"));
```

14. Toujours dans la classe `EtudiantsRepertoire`, créez une méthode `readAll()` qui renvoie la liste des étudiants du répertoire sous forme d'un objet de la classe `List<Etudiant>`.
15. Ajoutez l'attribut suivant à la classe `EtudiantService`.

```
private static EtudiantsRepertoire dao = new EtudiantsRepertoire();
```

16. Créez une méthode de la classe `EtudiantService` fournissant un document JSON contenant la liste de tous les étudiants présents dans la base via la requête REST suivante

```
GET /TP5Ex3/api/etudiant/tous HTTP/1.1
```

17. Dans la classe `EtudiantsRepertoire`, codez 4 méthodes `CRUD` permettant d'accéder aux données stockées dans ce répertoire. Elles auront comme signature respective :
    (a) Create : `public Etudiant create(Etudiant e)`
    (b) Read : `public Etudiant read(Integer numeroEtudiant)`
    (c) Update : `public Etudiant update(Etudiant e)`
    (d) Delete : `public void delete(int id)`
18. Ajoutez un constructeur vide à la classe `Etudiant`.
19. Ajoutez à la classe `EtudiantService` des méthodes permettant d'offrir les services suivants.
    (a) Create : `POST /TP5Ex3/api/etudiant/ HTTP/1.1`
    (b) Read : `GET /TP5Ex3/api/etudiant/find/12 HTTP/1.1`
    (c) Update : `POST /TP5Ex3/api/etudiant/update/15 HTTP/1.1`
    (d) Delete : `DELETE /TP5Ex3/api/etudiant/del/15 HTTP/1.1`
20. Testez avec votre navigateur la méthode de lecture.
21. Testez avec `curl` les accès en écriture avec une commande dans l'invite de commandes :

```
curl -X POST http://127.0.0.1:8080/TP5Ex3/api/Etudiant/
     -d "{\"anneeDeNaissance\": 2004,\"groupe\": \"SAGI2\",\"nom\":
↪  \"olive\",\"numeroEtudiant\": 9,\"prenom\": \"neuf\"}"
     -H "Content-Type: application/json"
```

Exercice 4 (Une application cliente de notre API)
L'objectif de cet exercice est de développer une application qui va "consommer" des données proposer par le service Web de l'exercice précédent.

1. Lisez le code suivant puis répondez aux questions suivantes
   (a) Quel est le rôle de la portion de codes des lignes 16-27 ?

(b) A quoi sert l'objet `ObjectMapper mapper` de la ligne 33 ?

(c) Quel est le but de l'instruction ligne 34 ?

```
1  package MonPack;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.net.HttpURLConnection;
7  import java.net.MalformedURLException;
8  import java.net.URL;
9
10 import org.codehaus.jackson.map.ObjectMapper;
11
12 public class Principale {
13
14   public static void main(String[] args) {
15     try {
16       URL url = new URL("http://localhost:8080/TP5Ex3/api/etudiant/id/1");
17       HttpURLConnection conn = (HttpURLConnection) url.openConnection();
18       conn.setRequestMethod("GET");
19       conn.setRequestProperty("Accept", "application/json");
20
21       if (conn.getResponseCode() != 200) {
22         throw new RuntimeException("Failed : HTTP error code : "
23         + conn.getResponseCode());
24       }
25
26       BufferedReader br = new BufferedReader(
27         new InputStreamReader((conn.getInputStream())));
28
29       String output;
30       System.out.println("Output from Server .... \n");
31       while ((output = br.readLine()) != null) {
32         System.out.println(output);
33         ObjectMapper mapper = new ObjectMapper();
34         Etudiant e = mapper.readValue(output, Etudiant.class);
35         System.out.println(e.toString());
36       }
37
38     }
39     catch (MalformedURLException e) {
40       e.printStackTrace();
41     }
42     catch (IOException e) {
43       e.printStackTrace();
44     }
45   }
```

```
46 }
```

2. Créez un projet Java (et non pas un projet dynamique Web).
3. Ajoutez un package `MonPack` et insérez la classe `Principale` du code précédent disponible via :
   `http://perso-laris.univ-angers.fr/~delanoue/polytech/jee/Principale.java`
   Note : vous aurez probablement besoin d'ajouter à votre Build Path les fichiers
   — `jackson-core-asl-1.9.13.jar`,
   — `jackson-mapper-asl-1.9.13.jar`.
4. Ajoutez à votre package la classe `Etudiant` de l'exercice .
5. A l'exécution, vous devriez pourvoir admirer les signaux suivants dans la console :

```
Output from Server ....
{"anneDeNaissance":1980,"groupe":"SAGI1","nom":"Skywalker1",
↪  "numeroEtudiant":1,"prenom":"Anakin"}
Etudiant [nom=Skywalker1, prenom=Anakin, anneDeNaissance=1980,
↪  numeroEtudiant=1, groupe=SAGI1]
```

Exercice 5

1. Créez une application Web Dynamique nommée `TP5EX5`.
2. Faites en sorte que votre application web affiche la liste des étudiants sur une page web accessible depuis un navigateur.
3. Proposez d'autres pages permettant de créer, lire, mettre à jour et supprimer (CRUD) les données fournies par l'API REST de l'exercice 4.