



Rapport JAX-RS

LO54 P16

Sibel bedir – Edourd Etienne – Vincent Merat

Table des matières

| | |
|--|----|
| 1. Définitions :..... | 3 |
| 2. Installation | 4 |
| 1. Tomcat | 4 |
| 2. NetBeans | 5 |
| 3. Créer un projet de web Application NetBeans | 7 |
| 4. Créer un web service RESTful | 9 |
| a. GET Service Handler | 9 |
| b. POST Service Handler | 12 |
| c. Déploiement du Web Service..... | 13 |
| d. Jersey Web Service Annotations | 13 |

Introduction :

Dans le cadre de l'uv LO54, nous devions apprendre à nous servir d'une technologie liée à Java EE. Ayant eu une première approche avec les web service durant notre stage, nous avons choisi d'étudier l'interface de programmation Java JAX-RS (Java API for RESTful Web Services), qui permet de créer des services Web avec une architecture REST.

Dans ce rapport nous allons présenter un tutorial sur l'utilisation de JAX-RS, puis pour conclure nous parlerons de notre expérience vis-à-vis de cette technologie.

1. Définitions :

REST : REpresentational State Transfer (REST) est un style d'architecture pour les systèmes hypermédia distribués comme le World Wide Web. Le centre de l'architecture RESTful est le concept de ressources identifiées par des identificateurs de ressources universel (URI).

Ces ressources peuvent être manipuler en manipulant une interface stantard (par exemple HTTP), les informations elles sont échangées en utilisant une représentation de ces ressources.

RESTful : Les services web RESTful sont des services utilisant le style d'architecture RESTful. Utiliser l'approche RESRfull pour créer des web services grâce à son habilité de transmettre des données directement, est une alternative de plus en plus populaire à SOAP.

2. Installation

Pour de pouvoir utiliser JAX-RS il faut tout d'abord configurer NetBeans et Tomcat afin qu'ils puissent fonctionner ensemble. Dans cette première partie nous allons donc expliquer comment les configurer mais aussi comment tester un web services avec un serveur Tomcat.

1. Tomcat

1. Télécharger et installer la version 8 d'Apache Tomcat.

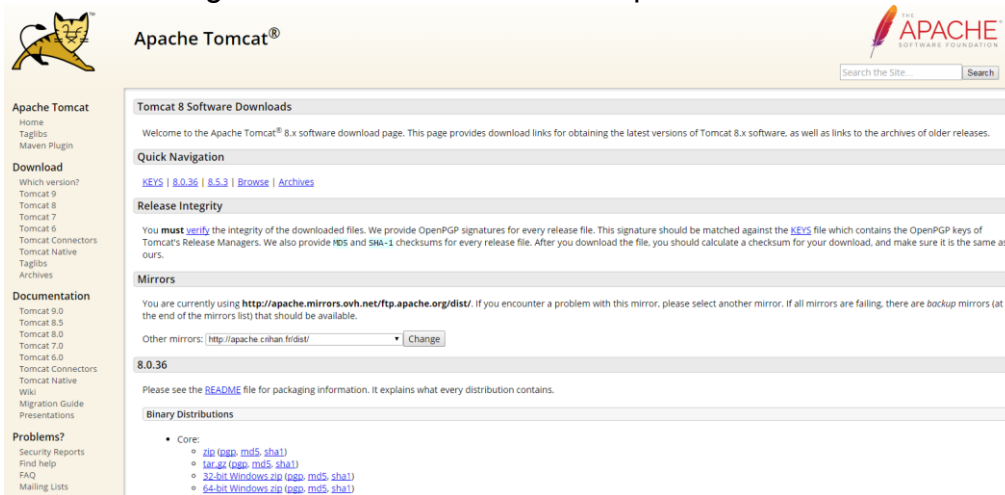


Figure 1. Page de téléchargement d'Apache Tomcat

2. IL faut à présent enregistrer un administrateur Tomcat, ce qui va permettre de lancer le système de management de Tomcat. Ajouter un nom d'utilisateur et un mot de passe (par exemple tomcat\tomcat). Pour cela il faut ouvrir le fichier *apache-tomcat-8.0.26/conf/tomcat-users.xml* .

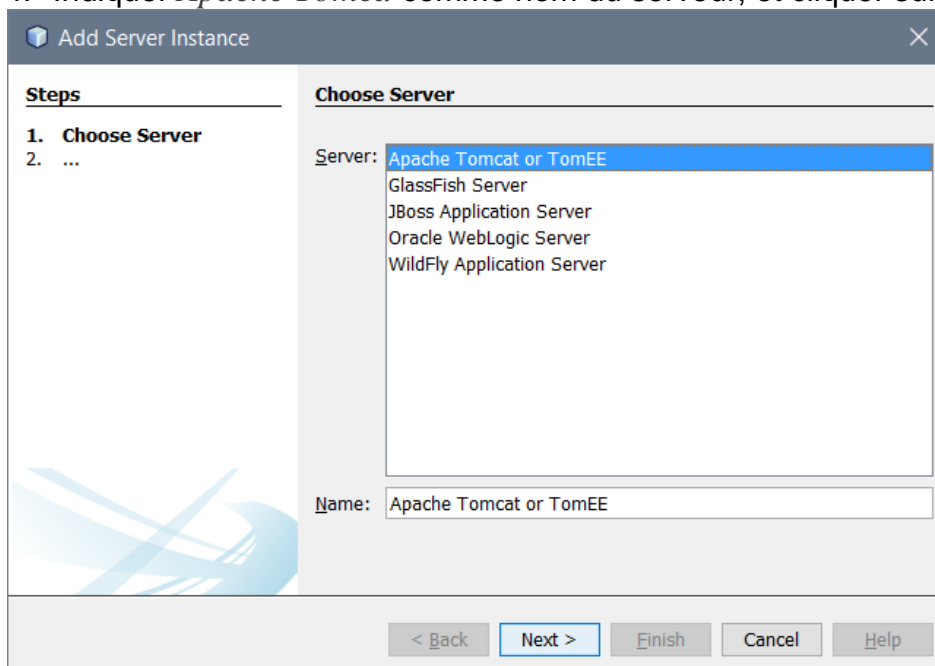
3. Commenter toutes les lignes et ajouter par le code suivant :

```
<xml version="1.0" encoding="UTF-8">
<tomcat-users>
  <role rolename="tomcat"/>
  <user username="tomcat" password="tomcat" roles="manager-script,manager-gui"/>
</tomcat-users>
```

4. Lancer ou arrêter le serveur Tomcat grâce aux lignes de commande suivantes :
`apache-tomcat-8.0.26/bin/startup.sh`
`apache-tomcat-8.0.26/bin/shutdown.sh`
5. Tester le serveur Tomcat en ouvrant sur votre navigateur l'URL *http://localhost:8080*.

2. NetBeans

1. Télécharger et installer la version Java EE de NetBeans. Download and install the Java EE version of NetBeans. Pour tester l'application web, NetBeans utilise le serveur web Glassfish par défaut. Pour utiliser Tomcat, cliquer sur l'onglet **Services** (pour faire apparaître l'onglet, sélectionner **Window > Services**).
2. Faire un clic droit sur **Servers** et sélectionner **Add Server**.
3. Dans la liste des serveurs, sélectionner **Apache Tomcat**.
4. Indiquer *Apache Tomcat* comme nom du serveur, et cliquer sur **Next**.



6. Dans l'onglet cliquer sur **Browse** dans l'onglet **Add Server Instance** pour indiquer la place du serveur.

Add Server Instance

Steps

1. Choose Server
2. **Installation and Login Details**

Installation and Login Details

Specify the Server Location (Catalina Home) and login details

Server Location:

☐ Use Private Configuration Folder (Catalina Base)

Catalina Base:

Enter the credentials of an existing user in the manager or manager-script role

Username:

Password:

☒ Create user if it does not exist

Tomcat with the same Catalina Home folder is already registered.

< Back Next > Finish Cancel Help

7. Sélectionner le dossier d'installation d'Apache Tomcat et cliquer sur **Open**.
8. Indiquer l'utilisateur Tomcat (suivant l'exemple précédent, nom : tomcat et password : tomcat), puis cliquer sur **Finish**. Apache Tomcat fais désormais parti des serveurs que vous pourrez sélectionner pour tester votre web service plus tard.

Add Server Instance

Steps

1. Choose Server
2. **Installation and Login Details**

Installation and Login Details

Specify the Server Location (Catalina Home) and login details

Server Location:

☐ Use Private Configuration Folder (Catalina Base)

Catalina Base:

Enter the credentials of an existing user in the manager or manager-script role

Username:

Password:

☒ Create user if it does not exist

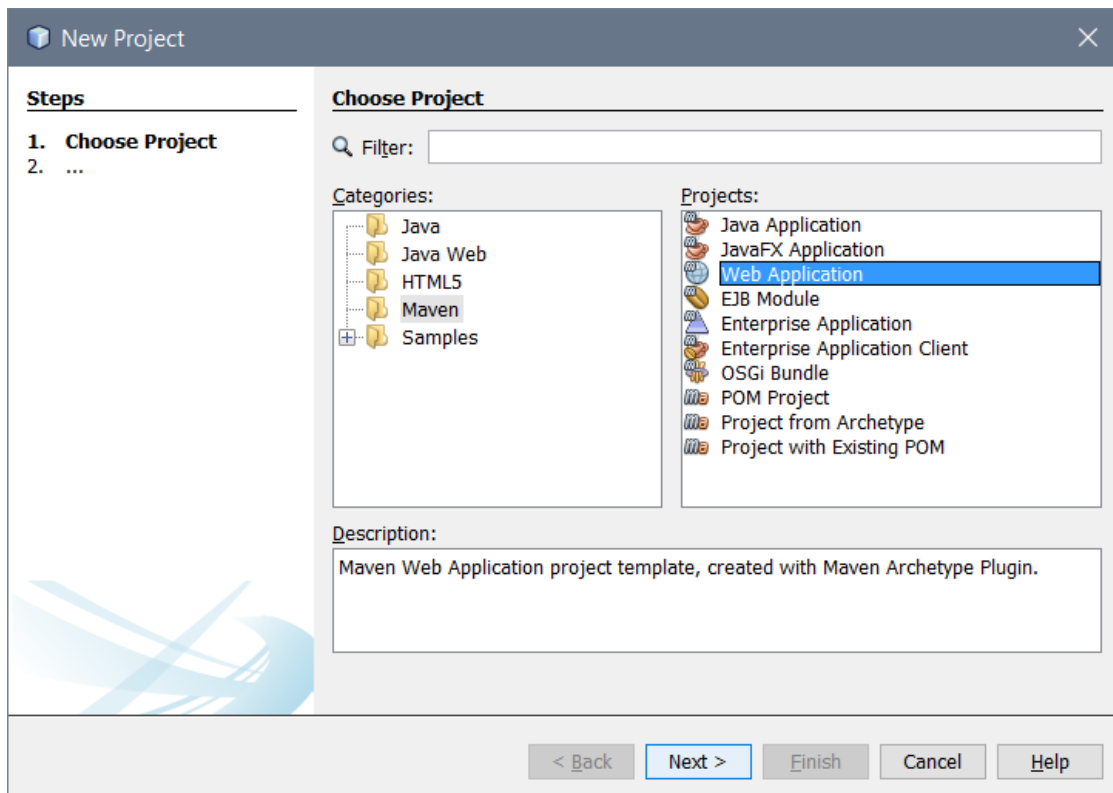
Tomcat with the same Catalina Home folder is already registered.

< Back Next > Finish Cancel Help

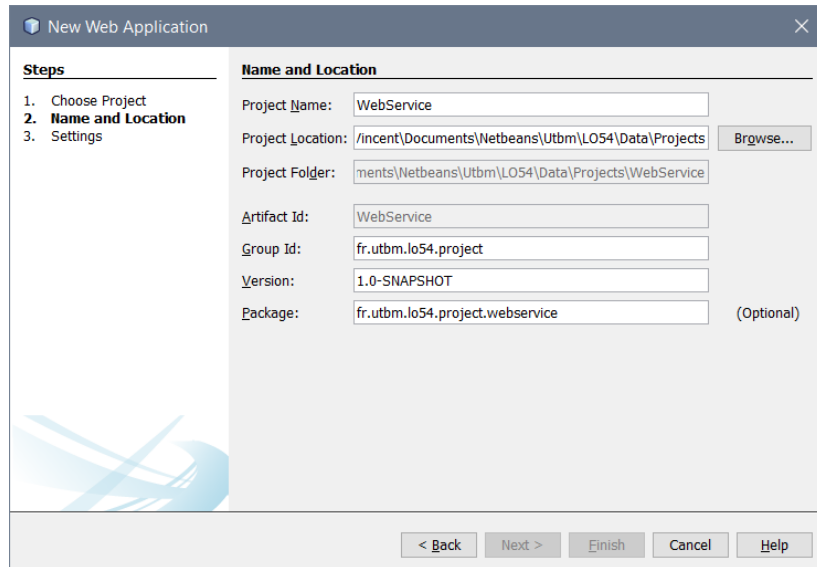
3. Créer un projet de web Application NetBeans

Après avoir configuré l'environnement de développement, vous devez tout d'abord construire une base d'application web à laquelle il faut ajouter un service RESTful et des méthodes.

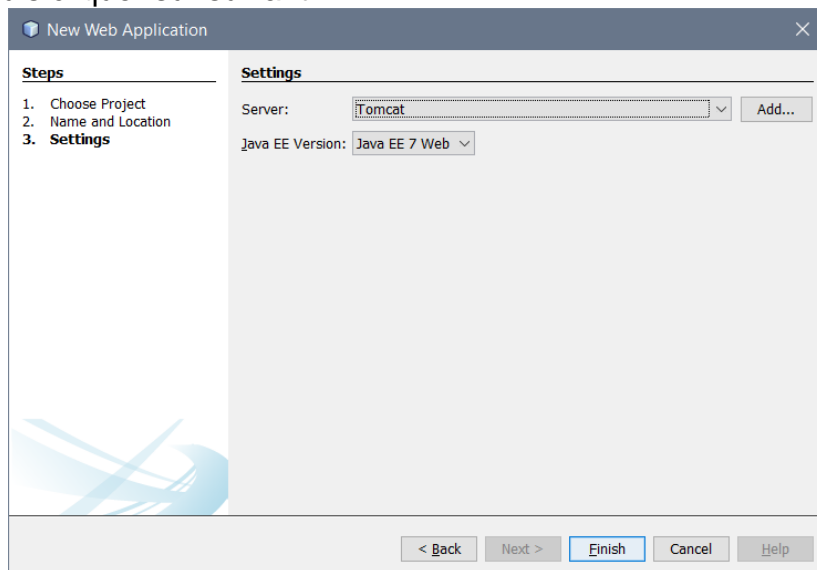
1. Cliquer sur **File > Open > New Project...**
2. Sélectionner la catégorie **Java Web** et le projet **Web Application**, puis cliquer sur suivant.



3. Entrer *getsomeres*t comme nom de projet et indiquer l'emplacement du dossier souhaité, puis cliquer sur suivant.



4. Dans l'onglet **Server and Settings**, sélectionner Apache Tomcat dans le champs **Server**, puis cliquer sur suivant.



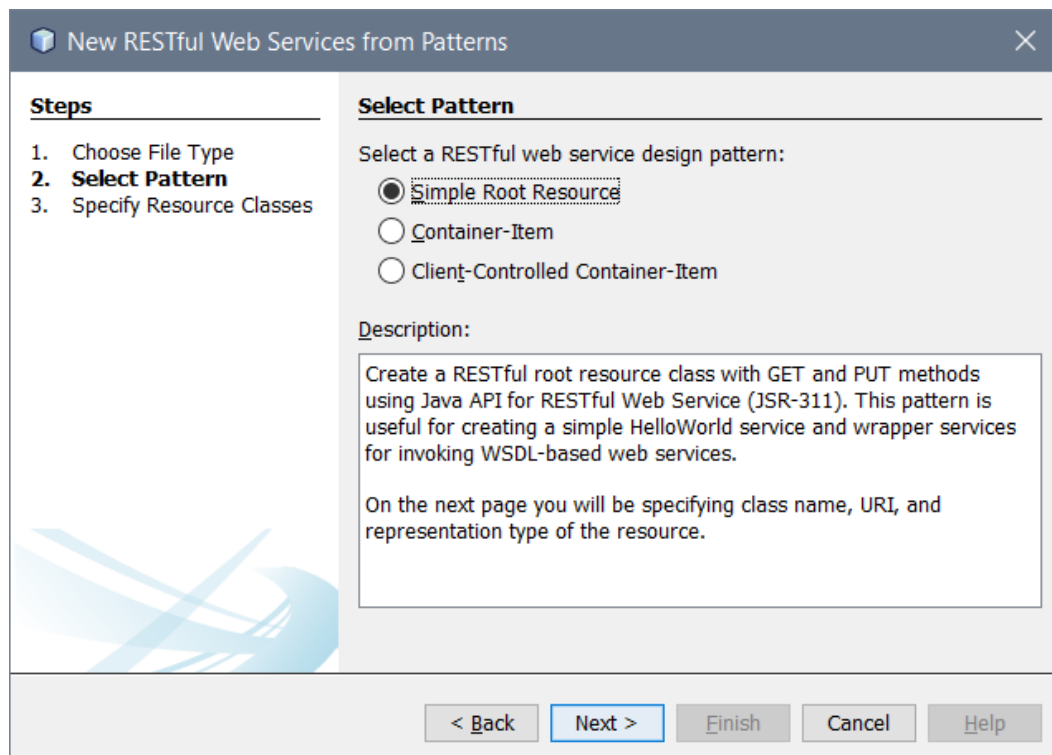
5. Nous ne voulons pas ajouter de frameworks supplémentaire, cliquer sur **Finish** dans l'onglet **Frameworks**.
6. Vous avez maintenant une application web très simple qui charge la page index.html. Pour lancer l'application, vous pouvez cliquer sur la flèche verte dans la barre d'outils de NetBeans, sélectionner **Run > Project** dans le menu ou bien appuyer sur **F6**.

4. Créer un web service RESTful

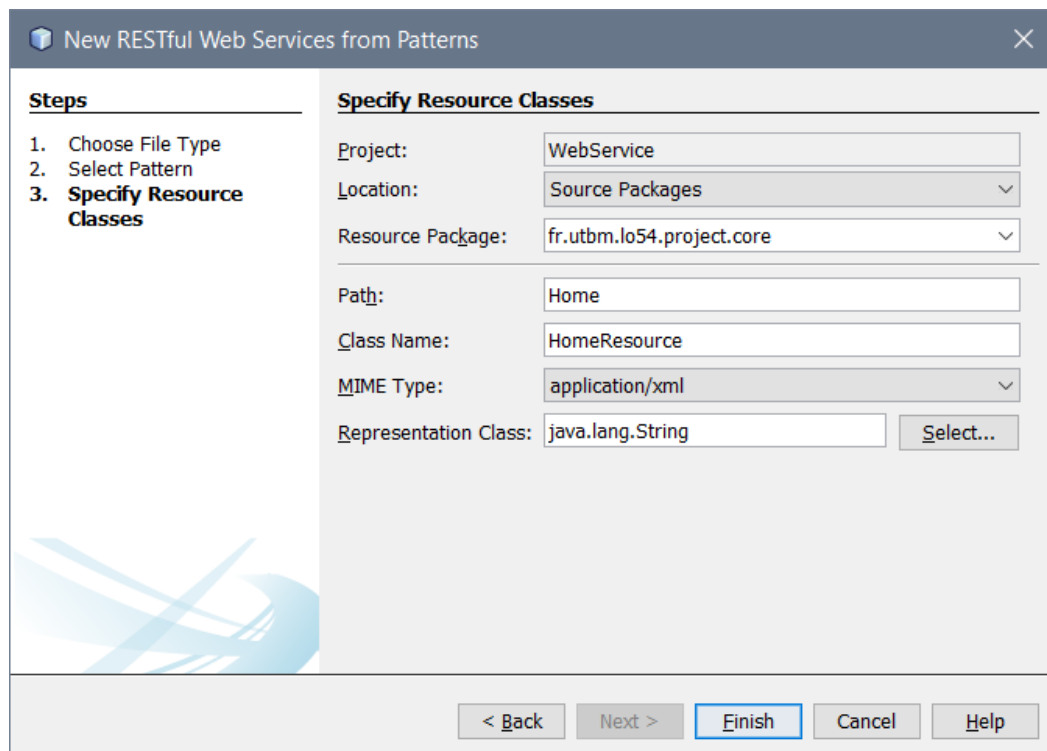
a. GET Service Handler

Maintenant que nous avons une applications web assez basic, nous allons ajouter un RESTful service qui répond à une requête HTTP GET.

1. Faite un clic droit sur getsomerest sur **Projects**.
2. Sélectionner **New...** puis **RESTful Web Services from Patterns...**
3. Sélectionner **Simple Root Resource** dans l'onglet **Select Pattern**, puis cliquer sur **Next**.



4. Vous devez maintenant spécifier les méthodes et les classe du service. Dans l'onglet **Specify Resource** indiquer le nom du package Java qui contient les classes du service.
5. Indiquer le chemin du service **Path** ainsi que le nom du service **Class name** dans *ServiceResource*.
6. Le type MIME fais référence au format du message que seras envoyer et retourner par notre web service. Indiquer le dans *text/html* pour un simple service GET.
7. Cliquer sur **Finish**.



The screenshot shows a dialog box titled "New RESTful Web Services from Patterns" with a close button (X) in the top right corner. On the left, a "Steps" panel lists three steps: 1. Choose File Type, 2. Select Pattern, and 3. Specify Resource Classes (which is currently selected). The main area is titled "Specify Resource Classes" and contains several input fields and dropdown menus:

- Project:** WebService
- Location:** Source Packages (dropdown menu)
- Resource Package:** fr.utbm.lo54.project.core (dropdown menu)
- Path:** Home
- Class Name:** HomeResource
- MIME Type:** application/xml (dropdown menu)
- Representation Class:** java.lang.String (text field) with a "Select..." button next to it.

At the bottom of the dialog, there are five buttons: "< Back", "Next >", "Finish" (highlighted with a blue border), "Cancel", and "Help".

8. Vous pouvez maintenant voir une **RESTful Web Services** dans votre projet. En l'ouvrant vous pouvez voir la classe services avec les methodes GET et PUT y. Remplacer le contenu de *getHtml()* avec la valeur à retourner.

```
1      package com.vichargrave;
2
3      import javax.ws.rs.core.Context;
4      import javax.ws.rs.core.UriInfo;
5      import javax.ws.rs.PathParam;
6      import javax.ws.rs.Consumes;
7      import javax.ws.rs.PUT;
8      import javax.ws.rs.Path;
9      import javax.ws.rs.GET;
10     import javax.ws.rs.Produces;
11
12     @Path("service")
13     public class ServiceResource {
14
15         @Context
16         private UriInfo context;
17
18         public ServiceResource() {
19         }
20
21         @GET
22         @Produces("text/html")
23         public String getHtml() {
24             return "<h1>Get some REST!</h1>";
25         }
26
27         @PUT
28         @Consumes("text/html")
29         public void putHtml(String content) {
30         }
31     }
```

9. Faites un clic droit sur le projet **getsomerest** et sélectionnez **Properties**, afin d'indiquer le chemin au web service de manière à ce qu'il soit pris automatiquement lors du test.

10. Sélectionnez **Run**.

11. Quand vous testez le web service, l'URL du GET doit être *behttp://localhost:8080/getsomerest/webresources/service* (la partie *getsomerest* de l'URL est définie dans l'attribut *path* du fichier *context.xml*, la partie *webresources* du chemin est indiqué dans le *@javax.ws.rs.ApplicationPath*(« *webresources* ») dans le fichier

ApplicationConfig.java créé par NetBeans. La partie service est définie par le `@Path` dans le fichier *ServiceResource.java*.

Entrer `/webresources/service` dans le champs **Relative URL**, puis cliquer sur **OK**.

12. La partie du chemin *webresources* est définie dans le fichier *ApplicationsConfig.java* file.

b. POST Service Handler

La plupart des web services ont un handler POST, nous allons donc en ajouter un. Le handler accepte le format texte depuis un REST client qui est spécifié par l'annotation Jersey `@Consumes("application/x-www-form-urlencoded")`. Dans ce cas le handler POST le texte décodé au client qui est spécifié par l'annotation Jersey `@Produces("text/plain")`.

1. Add the following import statement to the list of imports.

```
1 import javax.ws.rs.POST;
```

2. Ajouter le code ci-dessous à la classe *ServiceResource*. L'annotation `@Consumes` spécifie que le handler POST accepte les requêtes contenant des URL avec données codé. Le Content-Type de la requête POST doit correspondre au type `@Consumes`. L'annotation `@Produces` spécifie que le handler POST retourne du texte.

```
1 @POST
2 @Consumes("application/x-www-form-urlencoded")
3 @Produces("text/plain")
4 public String postHandler(String content) {
5     return content;
6 }
```

3. Lancer le serveur de test.
4. Vous pouvez tester le web service en lui envoyant un fichier depuis votre system

avec la commande **curl** suivante :

```
curl http://localhost:8080/getsomereset/webresources/service --data "Hello World!"
```

5. Le handler POST doit retourner *Hello World!* au **curl**.

c. Déploiement du Web Service

Maintenant que le web service RESTful avec les handlers GET et POST fonctionne, nous allons déployer le service vers notre serveur Tomcat. Les packages NetBeans sont tous dans un WAR package. Déployer le package de la façon suivante :

1. Ouvrir la page principale de Tomcat dans votre navigateur
2. Cliquer sur le bouton **Manager**.
3. Descendre sur **WAR file to deploy**.
4. Cliquer sur le bouton **Choose file**.
5. Aller jusqu'à votre dossier *NetBeansProjects/GetSomeRest/dist* et sélectionner *GetSomeRest.war*.
6. Cliquer sur **Deploy**. Dans la colonne **Path** sous **Applications** vous devez voir */GetSomeRest*.
7. Lancer la commande **curl** depuis la section précédente avec le host/IP et le port pour votre cible Tomcat, afin de vérifier le fonctionnement.

d. Jersey Web Service Annotations

Jersey fournit une collection d'annotation Java qui peuvent être utilisées pour définir la structure d'un web service. Vous pouvez retrouver les différentes annotations ainsi que leur méthode d'utilisation sur internet.

Conclusion

Nous avons pu nous rendre compte à travers l'utilisation de la technologie JAX-RS que son utilisation n'était pas très différente des méthodes vues en cours tout au long du semestre. Malgré notre appréhension sur cette technologie qui nous paraissait compliquée d'utilisation, de par notre manque de connaissance sur les web services et le manque de clarté des tutoriels trouvés, nous nous sommes rendu compte en avançant sur le projet de sa simplicité d'utilisation.

Par manque de temps nous n'avons pas pu explorer JAX-RS comme nous l'aurions voulu afin de l'intégrer dans le projet qui nous avait été demandé de faire. Mais la mise en place de ce tutoriel et les tests que nous avons effectués, nous a permis de nous familiariser et d'appréhender une nouvelle façon de créer des web services.