# Etude d’un Web Service

# Rest Ful sous Tomcat

Introduction

Vous disposez d’une machine virtuelle sur laquelle sont installées les applications suivantes :

* Eclipse Luna
* Xampp avec une base de données nommée baseoeuvre
* Apache Tomcat

## Objectifs de ce TP

Les objectifs de ce travail pratique vous permettront de comprendre l’architecture d’un Web Service Rest Ful réalisé sous le serveur Tomcat et de sa consommation par un Client écrit avec une architecture MVC.

## Présentation

### Web Service / Maven

Voici les composants de ce web service :

* + d’une couche service
    - classe WSservice qui offre les ressources
    - Classe WSApplication pour la création
  + d’une couche métier
    - classes java issues de la base de données
  + D’une couche d’infrastructure (persistance)
    - Classe Connexion
    - Classe DbDialogue

### Client / Maven

* **L’architecture MVC (Modèle-Vue-Contrôle)** d’une application web
  + Couche présentation
    - pages JSP, langage jstl
  + Couche consommation
    - Deux classes génériques d’appel des ressources
  + d’une couche métier
    - classes java issues de la base de données
  + D’une couche Contrôleur
    - Classe Contrôleur
  + D’une couche d’infrastructure (persistance)
    - Classe Connexion
    - Classe DbDialogue

### Architecture du web service et du client

|  |  |
| --- | --- |
|  |  |

## Déploiement de l’application

L’environnement de développement est :

* Intellij 2017.1
* Tomcat 9.0 sous java : JDK1.8
* Mysql 6.0 sous Xampp
* Driver mysql-connector-java-5.2.XX-bin (déployé sous C:\apache-tomcat-8.0.17\lib

# Travail à faire

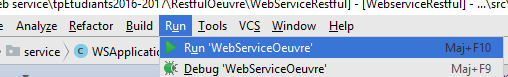
## Télécharger les applications

## 

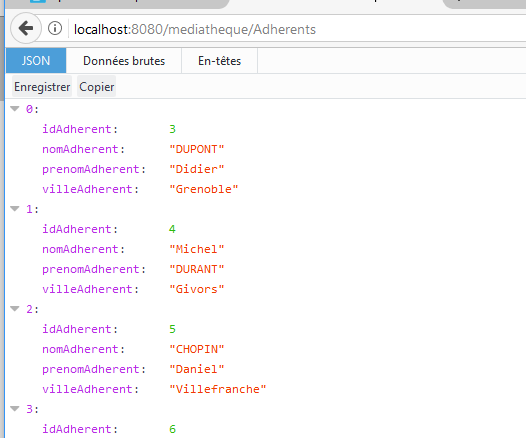
* Web service RestFull
* Client Rest Full
* La base œuvre

## Lancement de l’application

Vous pouvez lancer le Web Service en se plaçant sur le projet puis bouton droit

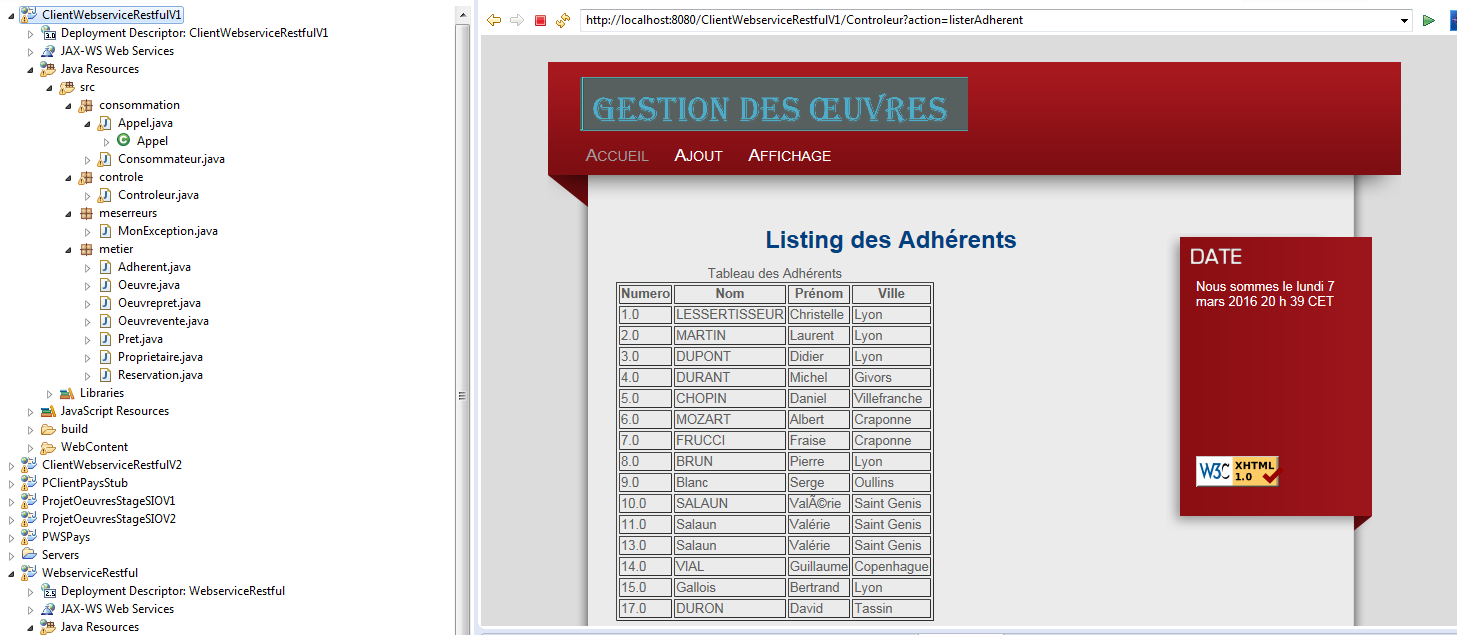


Un flux json est alors affiché dans le navigateur.



Un flux json est alors affiché dans le navigateur.

Le client se lance de la même manière mais **vous devez trouver une solution pour accéder au web service généré par Intellij.**



Seuls les items Affichage et Ajout sont accessibles. Vous pouvez les tester :

|  |  |
| --- | --- |
|  |  |

## Etude de l’application

### Répondez aux questions suivantes :

1. Comment le client appelle-t-il une ressource ? Illustrez votre réponse par l’appel de la liste des adhérents ?

|  |
| --- |
| Le client effectue une requête http sur le service RestFul.  String ressource = **"/Adherents"**;  Appel unAppel = **new** Appel(); reponse = unAppel.appelJson(ressource); Gson gson = **new** Gson(); List<Adherent> json = gson.fromJson(reponse, List.**class**); |

1. Dans le contrôleur du client, que contient la variable ressource ?

|  |
| --- |
| La variable ressource contient le chemin relatif vers la ressource recherchée.  String ressource = **"/Adherents"**; |

1. Dans le web service quelle est la méthode qui transforme les données ( collection, objet) sous forme de flux json  ?

|  |
| --- |
| La méthode qui transforme les données sous forme json est :  **public** String Gson::toJson(Object src) |

1. Quel est le rôle de la classe consommateur.java?

|  |
| --- |
| La classe Consommateur permet de générer un client et permet d’effectuer les requêtes http au service. |

1. Quelle est le rôle de la classe WService.java  ?

|  |
| --- |
| La classe WService définit et implémente les différentes fonctionnalités du service (ressources). |

## Appel d’un nouveau service

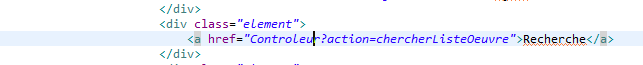
Nous allons ajouter l’appel d’un nouveau service présent dans le web service : recherche d’une œuvre.





### Couche de présentation

Nous ajoutons une nouvelle action dans la partie menu de notre page d’accueil. Cette action nous permettra d’alimenter la liste déroulante pour le choix d’une œuvre.



### Couche Contrôleur

Nous ajoutons la constante suivante ***RECHERCHER\_LISTE\_OEUVRE***



Le traitement correspondant dans le contrôleur est :

**else** **if** (***RECHERCHER\_LISTE\_OEUVRE***.equals(actionName)) {

String ressource = "/Oeuvres";

**try** {

Appel unAppel = **new** Appel();

reponse = unAppel.appelJson(ressource);

Gson gson = **new** Gson();

List<Oeuvrevente> json = gson.fromJson(reponse, List.**class**);

request.setAttribute("mesOeuvres", json);

destinationPage = "/rechercherOeuvre.jsp";

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

destinationPage = "/index.jsp";

request.setAttribute("MesErreurs", e.getMessage());

}

}

Ce traitement fait appel à une ressource consulterListeOeuvre () présente dans le web service

### rechercherOeuvre.jsp

Cette page va afficher une liste déroulante pour le choix d’une œuvre :

Voici le code de cette page

<%@ include file=*"include/header.jsp"*%>

<body>

<div id=*"site"*>

<%@ include file=*"include/menu.jsp"*%>

<div id=*"conteneur"*>

<%@ include file=*"include/bandeaudroite.jsp"*%>

<div id=*"contenu"*>

<P align=*"center"*>

<FONT face=*"Arial"* size=*"5"* color=*"#004080"*> <STRONG>Recherche &nbsp;

d'une oeuvre </STRONG></FONT>

</P>

<h1>Recherche d'une oeuvre </h1>

<h2>A l'aide d'une liste</h2>

<form method=*"post"* action=*"Controleur?action=rechercherOeuvre"*>

<div>

<select name=*"id"*>

<c:forEach var=*"uneOeuvre"* items=*"*${mesOeuvres}*"*>

<option value=*"*${uneOeuvre.idOeuvrevente}*"*> ${uneOeuvre.titreOeuvrevente}

</option>

</c:forEach>

</select>

<input type=*"submit"* value=*"Rechercher l'oeuvre "* />

</div>

</form>

</div>

<%@ include file=*"include/footer.jsp"*%>

</div>

</div>

</body>

</html>

Cette page appelle le contrôleur pour invoquer un traitement de recherche d’une œuvre. Voici le code à insérer :

**else** **if** (***RECHERCHER\_OEUVRE***.equals(actionName)) {

Oeuvrevente uneOeuvre = **null**;

**if** (request.getParameter("id") != **null**) {

**try** {

// Oeuvres/{Id}")

Gson gson = **new** Gson();

**int** idoeuvre = gson.fromJson(request.getParameter("id"), Integer.**class**);

String ressource = "/Oeuvres/" + idoeuvre;

Appel unAppel = **new** Appel();

reponse = unAppel.appelJson(ressource);

Oeuvrevente json = gson.fromJson(reponse, Oeuvrevente.**class**);

request.setAttribute("uneOeuvre", json);

destinationPage = "/afficherOeuvre.jsp";

} **catch** (Exception e) {

destinationPage = "/erreur.jsp";

request.setAttribute("MesErreurs", e.getMessage());

}

}

}

On note que le contrôleur appelle la ressource /Œuvre/ avec l’id d’une œuvre, présente dans le web service.

Enfin voici le code la page qui affiche les informations recherchées.

<%@ include file=*"include/header.jsp"*%>

<body>

<div id=*"site"*>

<%@ include file=*"include/menu.jsp"*%>

<div id=*"conteneur"*>

<%@ include file=*"include/bandeaudroite.jsp"*%>

<div id=*"contenu"*>

<P align=*"center"*>

<FONT face=*"Arial"* size=*"5"* color=*"#004080"*> <STRONG>Recherche &nbsp;

d'une oeuvre </STRONG></FONT>

</P>

<h1>Affichage d'une oeuvre </h1>

<ul>

<li><b>ID :</b> ${uneOeuvre.idOeuvrevente}</li>

<li><b>Titre :</b> ${uneOeuvre.titreOeuvrevente}</li>

<li><b>Etat :</b> ${uneOeuvre.etatOeuvrevente}</li>

<li><b>Prix :</b> ${uneOeuvre.prixOeuvrevente}</li>

<li><b>Nom Proriétaire :</b> ${uneOeuvre.proprietaire.nomProprietaire}</li>

</ul>

</div>

<%@ include file=*"include/footer.jsp"*%>

</div>

</div>

</body>

</html>

1. Mettre en place **les requêtes avec POST (création - modification) et Delete** pour les œuvres

Déposez sur Spiral : tp.docx – client (compressé) – web service (compressé)