Polytech 5A

RESTFUL

Christian VIAL

WEB SERVICE

- **Une introduction aux services Web RESTful**

REST est l'acronyme d'un style architectural, de représentation State Transfer.
Il s'agit d'un style architectural pour la construction de systèmes logiciels distribués.
Il est inspiré de la façon dont les données sont représentées, accessibles et modifiables sur le Web. .

Architecture de Rest

Ressource

URL + les données

Principe de base

ressources-oriented architecture.

```
WebResource webResource = client.resource(BASE_URI+"?pnom=Mathilde");
```

WEB SERVICE

- **Méthodes HTTP**
- **Utilisation du protocole HTTP**

• **GET**	**Get** est utilisé pour récupérer des données ou effectuer une recherche sur une ressource. Les données retournées par le service Web est une représentation de la ressource demandée.
• **POST**	**POST** est utilisé pour mettre à jour les données à l'aide d'une ressource. Le service web peut répondre avec les données ou le statut indiquant le succès ou l'échec.
• **PUT**	**PUT** est utilisé pour créer les données à l'aide d'une ressource.
• **DELETE**	**DELETE** est utilisé pour supprimer les données à l'aide d'une ressource.

WEB SERVICE

Formats de sortie

RESTful web services peuvent être programmés de telle manière qu'elles servent de données de **divers mime-types**. Par exemple, ils pourraient prendre et de répondre aux demandes de types de données suivantes en fonction de l'acceptation de type de la requête:

Plain-Text (text/plain)
format de sortie le plus simple : texte

HTML (text/html)
Format de sortie Html

JavaScript Object Notation (application/json)
Format de document Json
(ensembles de paires nom / valeur ou listes ordonnées de valeurs.)

XML (application/xml)
sortie sous la forme Xml

WEB SERVICE

- **Formats de sortie : Json**

Plain-Text (text/plain)

```
@GET
@Path("/getjson")
@Produces("application/json")
@Produces("text/plain")
// http://localhost:8080/demo/ressources/getjson

public Etudiant getEtudiantToJSON() throws ParseException {

    Etudiant unEtudiant = new Etudiant();
    unEtudiant.setNom("Vial");
    unEtudiant.setPrenom("Antoine");
    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
    Date unedate = sdf.parse("22/09/1988");
    unEtudiant.setDnaissance(unedate);
    return unEtudiant;
}
```

WEB SERVICE

- **Formats de sortie : les exemples**

Plain-Text (text/plain)

```
@GET
@Path("test/{Id}")
  // récupère la valeur passéé par  webResource.path("test").path(« xxxxx")
@Produces("text/plain")
  // http://localhost:8080/demo/ressources/test/christian

 public String donneunautrebonjour(@PathParam("Id") String id)
        throws Exception {
      if ( id != null ) {
         return "Bonjour   " + id + " et bienvenue dans le monde
RestFul!";
      }
      return "Bienvenue  id dans le monde RestFul!";
   }
```

WEB SERVICE

- **Formats de sortie : les exemples**

Plain-Text (text/plain)

```
@GET
 @Produces("text/plain")
   // http:///localhost:8080/demo/ressources?pnom=christian

   // QueryParam  représente un paramètre et non un path
 public String salutation(@QueryParam("pnom") String pnom) {
    if ( pnom != null ) {
      return "Bonjour " + pnom + " et bienvenue dans le monde Restful!";
        }
       return "Bienvenue pnom (queryparam)  dans le monde RestFul";
     }
```

WEB SERVICE

- **Formats de sortie : les exemples**

Plain-Text (text/plain)

```
@GET
 @Produces("text/plain")
   // http://localhost:8080/demo/ressources?pnom=christian

    // QueryParam  représente un paramètre et non un path
 public String salutation(@QueryParam("pnom") String pnom) {
    if ( pnom != null ) {
      return "Bonjour " + pnom + " et bienvenue dans le monde
Restful!";
        }
        return "Bienvenue pnom (queryparam)  dans le monde RestFul";
     }
```

WEB SERVICE

- **Formats de sortie : les exemples**

Plain-Text (text/html)

```
@GET
  @Path("html/{Id}")
// récupère la valeur passéé par WebResource.path("html")
  @Produces("text/html")
  // http://localhost:8080/demo/ressources/html/christian
  public String donnebonjourhtml(@PathParam("Id") String id) throws
Exception {
      if ( id != null ) {
        return "<html><body><h1>Bonjour   " + id + " et bienvenue dans
le monde RestFul! </body></h1></html>";
      }
      return "Bienvenue  id dans le monde RestFul!";
  }
```

WEB SERVICE

- **Formats de sortie : les exemples**

Plain-Text (application/xml)

```
@GET
@Path("/get")
@Produces("application/xml")
public Etudiant getEtudiantToXML()
    throws ParseException {
    Etudiant unEtudiant = new Etudiant();
    unEtudiant.setNom("Vial");
    unEtudiant.setPrenom("Antoine");

     SimpleDateFormat sdf =  new SimpleDateFormat("dd/MM/yyyy");
      Date unedate = sdf.parse("22/09/1988");
       unEtudiant.setDnaissance(unedate);
      return unEtudiant;
  } // http://localhost:8080/demo/ressources/get
```

```
@XmlRootElement(name = "Etudiant")
public class Etudiant {

String nom;
String prenom;
Date dnaissance;


public Etudiant() {
}

@XmlElement
public String getNom() {
return nom;
}
```

WEB SERVICE

- **RESTful Web Services vs SOAP Web Services**

RESTful	SOAP
ROA - Resource oriented architecture	SOA - SOAP oriented architecture
La Réponse est encapsulée dans une enveloppe HTTP.	La Réponse est encapsulée dans une enveloppe SOAP, qui est elle-même dans une enveloppe HTTP
WADL décrit le web service.	WSDL décrit le web service.
Léger facile à developer.	Lourd, complexe à développer.
Lié au protocole HTTP	Incorporé dans l'appel RPC

Développement d'une Application

Tomcat 9.0

Intellij

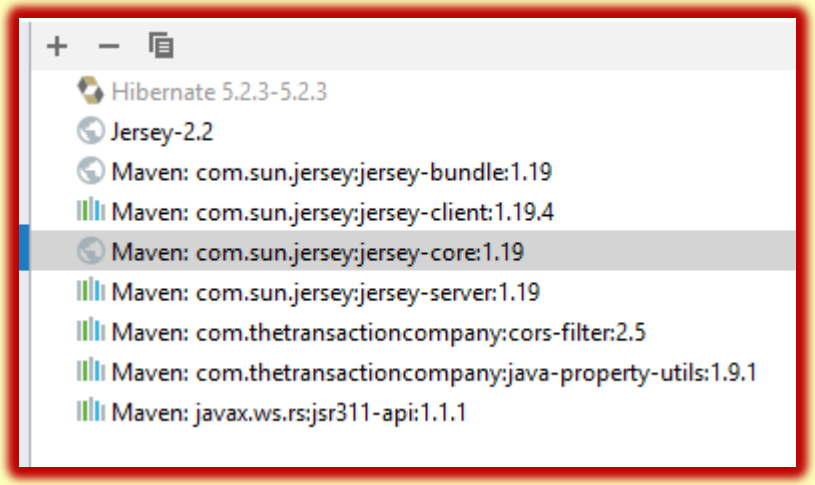
WEB SERVICE

Environnement de développement

- Tomcat 9.0
- Intellij IDE for Web Developers.

- Librairies : Pour le client consommateur

-Web Service : Bibliothèques

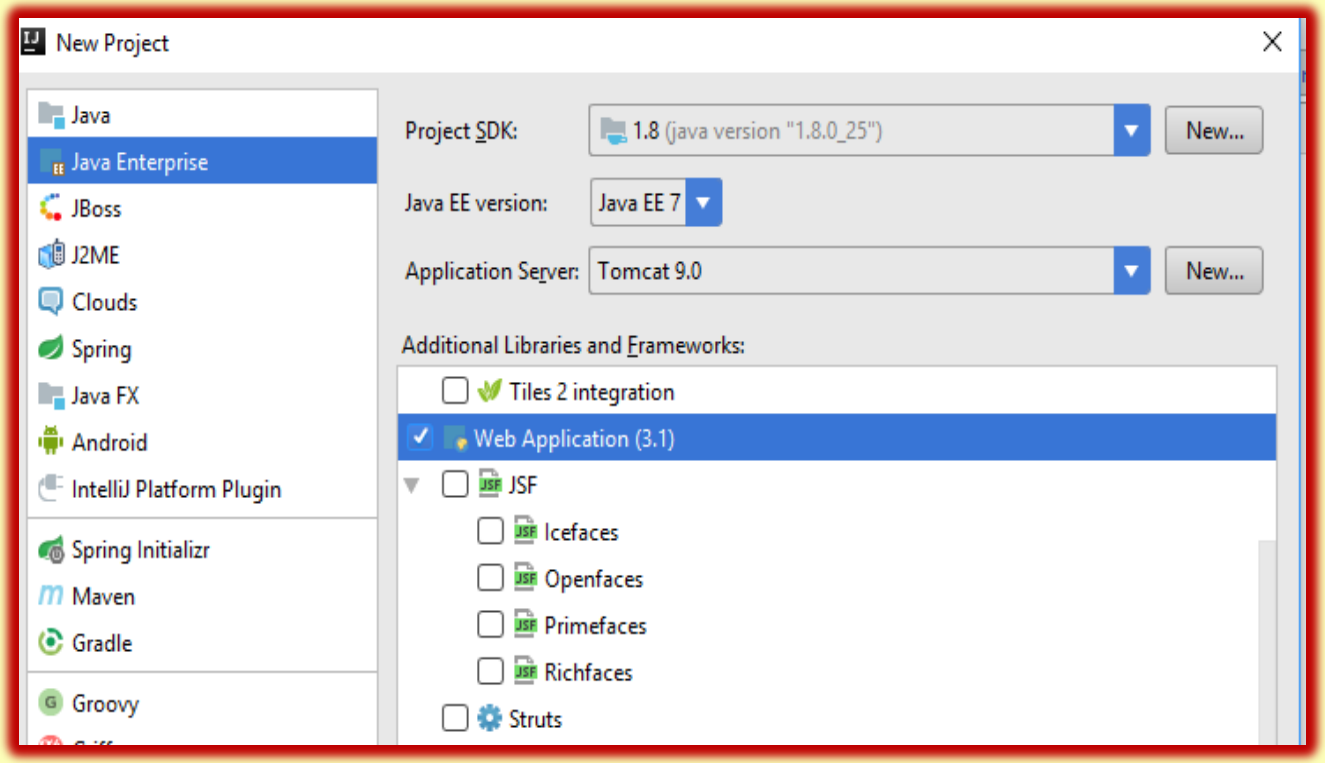



```
<dependency>
    <groupId>javax.ws.rs</groupId>
    <artifactId>javax.ws.rs-api</artifactId>
    <version>2.0.1</version>
    <scope>provided</scope>
</dependency>
```

WEB SERVICE

- **Développement Web Service ProjetRestTomcat**

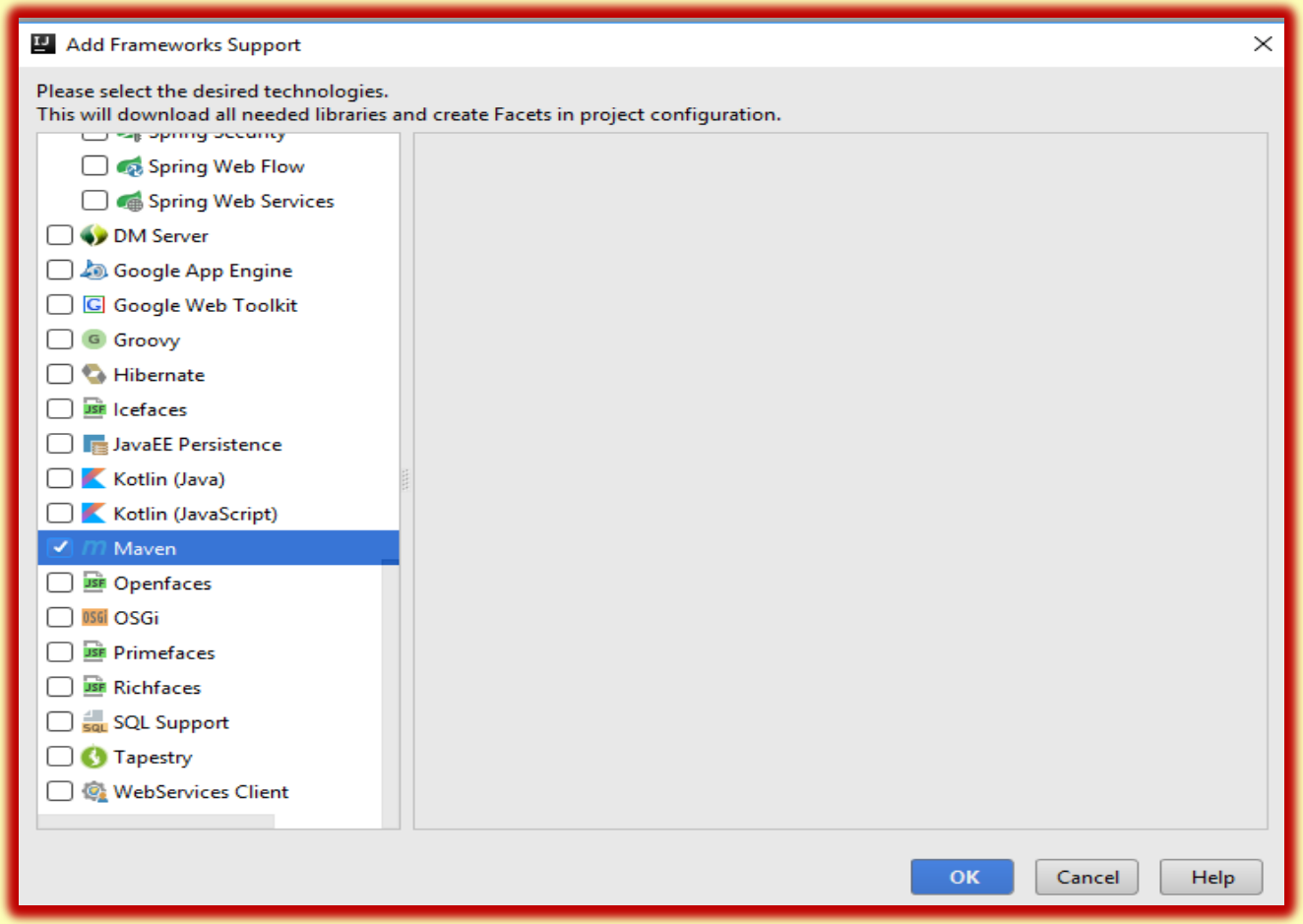
-



WEB SERVICE

Développement Web Service

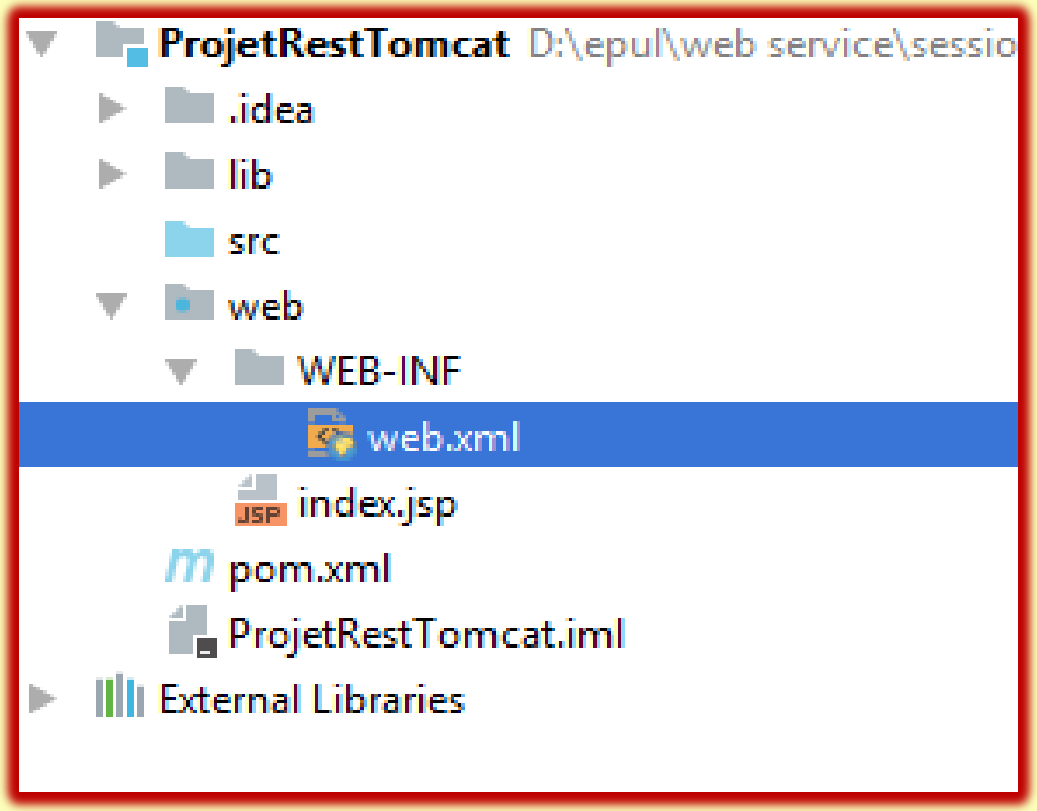
- Maven



WEB SERVICE

Développement Web Service

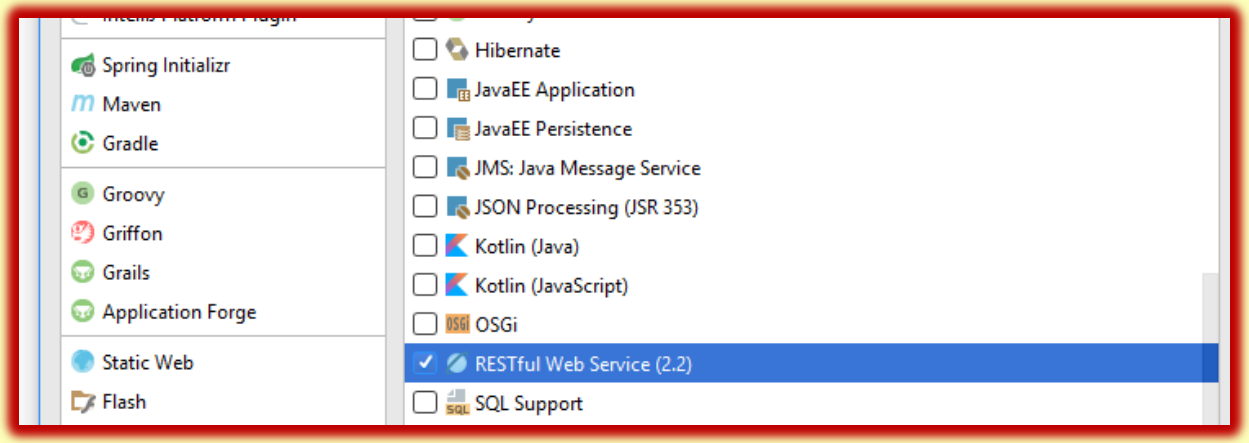
- Architecture du projet



WEB SERVICE

- **Développement Web Service**

- RESTful Web Service



Créez le Web Service : ProjetRestTomcat

WEB SERVICE

- **Maven dépendances**

```
<dependencies>

    <!-- jax-rs -->
    <dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-bundle</artifactId>
        <version>1.19</version>
    </dependency>
    <dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-server</artifactId>
        <version>1.19.4</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-core -->
    <dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-core</artifactId>
        <version>1.19.4</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-client -->
    <dependency>
        <groupId>com.sun.jersey</groupId>
        <artifactId>jersey-client</artifactId>
        <version>1.19.4</version>
    </dependency>
    <!-- corsfilter -->
    <dependency>
        <groupId>com.thetransactioncompany</groupId>
        <artifactId>cors-filter</artifactId>
        <version>2.5</version>
    </dependency>

</dependencies>
```

WEB SERVICE

- **Développement Web Service**

- Le projet doit contenir un singleton

```java
import java.util.Set;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

// addresse de basse pour accéder au WS
@ApplicationPath("/")
public class RestFulApplication extends Application {
    private Set<Object> singletons = new HashSet<Object>();
    private Set<Class<?>> empty = new HashSet<Class<?>>();

    public RestFulApplication() {
        singletons.add(new WSRessource());
    }

    @Override
    public Set<Class<?>> getClasses() {
        return empty;
    }

    @Override
    public Set<Object> getSingletons() {
        return singletons;
    }
}
```

Nom du web service

WEB SERVICE

Classe WSRessource

C'est la classe de notre WebService, elle va contenir les ressources, son entête commence par déclarer un path sur ressources, donc notre uri aura la forme
localhost :8080/…/ressources/….

```
package ws;

import javax.ws.rs.*;
import metier.Etudiant;
import java.util.*;
import java.text.*;

@Path("/ressources")
public class WSRessource {
    //@Context
    //private UriInfo context;

    /** Creates a new instance of WsSalutation */
    public WSRessource() {
    }
```

WEB SERVICE

Ressource de salutation en plain/text

Cette ressource attend un paramètre et affiche du texte

```
   @GET
   @Path("/hello/{unnom}")
   // récupère la valeur passéé par webResource.path("hello").path("xxxx")
   @Produces("text/plain")
   // http://localhost:8080/demo/ressources/hello/chistian
   public String donneBonjour(@PathParam("unnom") String name) {
      if (name != null) {
         return "Bonjour  " + name + " et bienvenue dans le monde
RestFull!";
      }
      return "Bienvenue xxxxxx  dans le monde RestFul!";
   }
```

WEB SERVICE

Ressource XML

On produit des informations au format xml

```
// /
// On récupère un objet sous la forme XML
// /
@GET
@Path("/get")
@Produces("application/xml")
// http://localhost:8080/demo/ressources/get
public Etudiant getEtudiantToXML() throws ParseException {

   Etudiant unEtudiant = new Etudiant();
   unEtudiant.setNom("Vial");
   unEtudiant.setPrenom("Antoine");

   SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
   Date unedate = sdf.parse("22/09/1988");
   unEtudiant.setDnaissance(unedate);
   return unEtudiant;
}
```

WEB SERVICE

Ressource Json

La ressource va produire les données au format json, c'est le format le plus **employé**.

```
@GET
@Path("/getjson")
@Produces("application/json")
// http://localhost:8080/demo/ressources/getjson
public Etudiant getEtudiantToJSON() throws ParseException {

    Etudiant unEtudiant = new Etudiant();
    unEtudiant.setNom("Vial");
    unEtudiant.setPrenom("Antoine");

    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
    Date unedate = sdf.parse("22/09/1988");
    unEtudiant.setDnaissance(unedate);
    return unEtudiant;

}
```

WEB SERVICE

Ressource HTML

La ressource renvoie les données au format HTML.

```
   //
   // Format de sortie : Html
   @GET
   @Path("html/{Id}")
   // récupère la valeur passéé par webResource.path("html").path("xxxx")
   @Produces("text/html")
   // http://localhost:8080/demo/ressources/html/christian
   public String SalutationHtml(@PathParam("Id") String id) throws
Exception {
      if (id != null) {
         return "<html><body><h1>Bonjour   "
               + id
               + " et bienvenue dans le monde RestFul!
</body></h1></html>";
      }
      return "Bienvenue  id dans le monde RestFul!";
   }
}
```

WEB SERVICE

Ressource avec un paramètre de type ?

La ressource reçoit un paramètre de type ?para=value.

```
//
// Appel du paramètre avec la notation ?
// /
@GET
@Produces("text/plain")
// http://localhost:8080/demo/ressources?pnom=christian
public String salutationParametree(@QueryParam("pnom") String pnom) {
   if (pnom != null) {
      return "Bonjour(QueryParam) " + pnom
            + " et bienvenue dans le monde Restful!";
   }
   return "Bienvenue pnom (queryparam)  dans le monde RestFul";
}
```

WEB SERVICE

Web.xml

Ce fichier va nous permettre de définir :

- Le chemin de la ressource
- Les modes acceptés (get, post…)
- Les filtres

Chemin du package de la classe ressource

- param-name permet de définir où trouver les ressources.
- url-pattern définit le format de l'uri : localhost :8080/**demo**/

```
<servlet>
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>ws</param-value>
    </init-param>

    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-lass>
        com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/demo/*</url-pattern>
</servlet-mapping>
```

WEB SERVICE

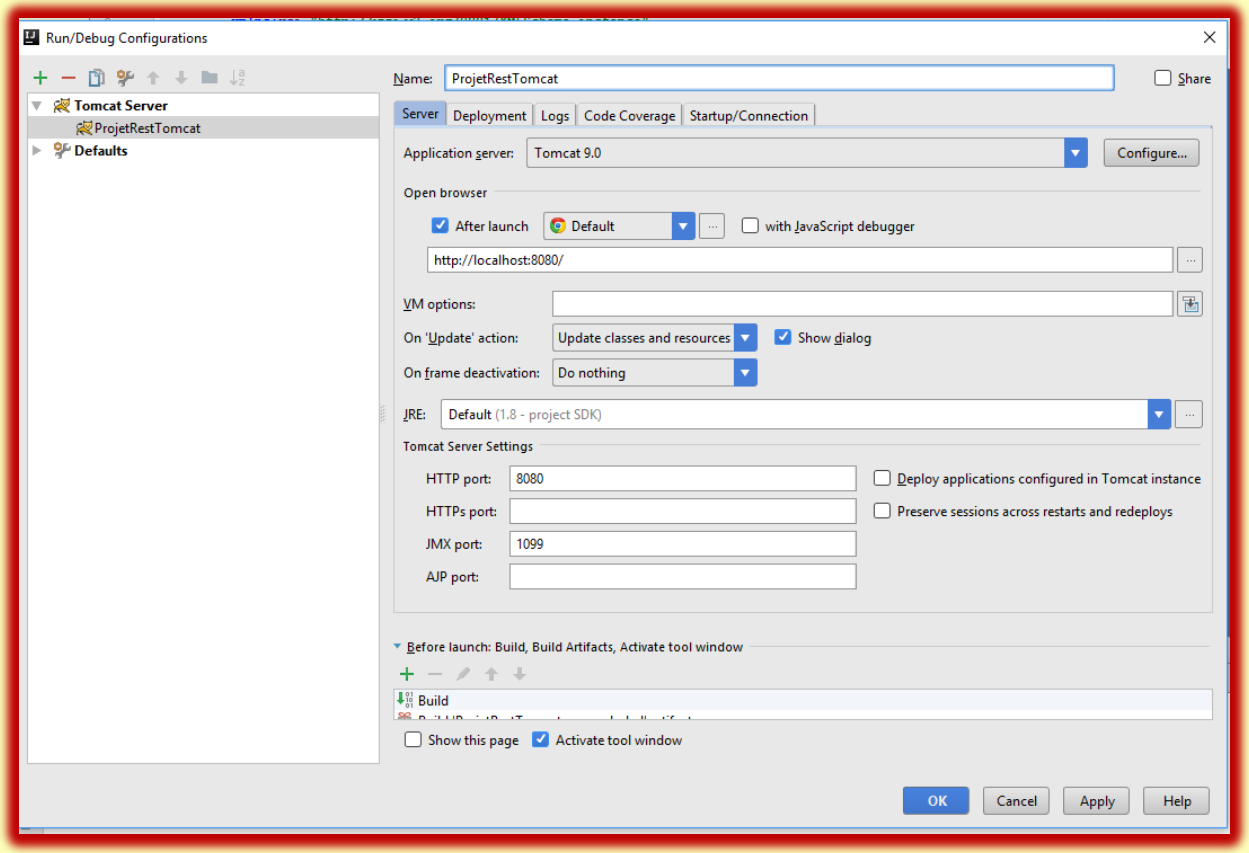
Filtres acceptés

Les balises suivantes permettent de définir les filtres qui seront acceptés.

```
<filter>
    <filter-name>CORS</filter-name>
    <filter-class>com.thetransactioncompany.cors.CORSFilter</filter-class>
    <init-param>
        <param-name>cors.allowOrigin</param-name>
        <param-value>*</param-value>
    </init-param>
    <init-param>
        <param-name>cors.supportedMethods</param-name>
        <param-value>GET, POST, HEAD, PUT, DELETE</param-value>
    </init-param>
    <init-param>
        <param-name>cors.supportedHeaders</param-name>
        <param-value>Accept, Origin, X-Requested-With, Content-Type, Last-
Modified</param-value>
    </init-param>
    <init-param>
        <param-name>cors.exposedHeaders</param-name>
        <param-value>Set-Cookie</param-value>
    </init-param>
    <init-param>
        <param-name>cors.supportsCredentials</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>CORS</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

WEB SERVICE

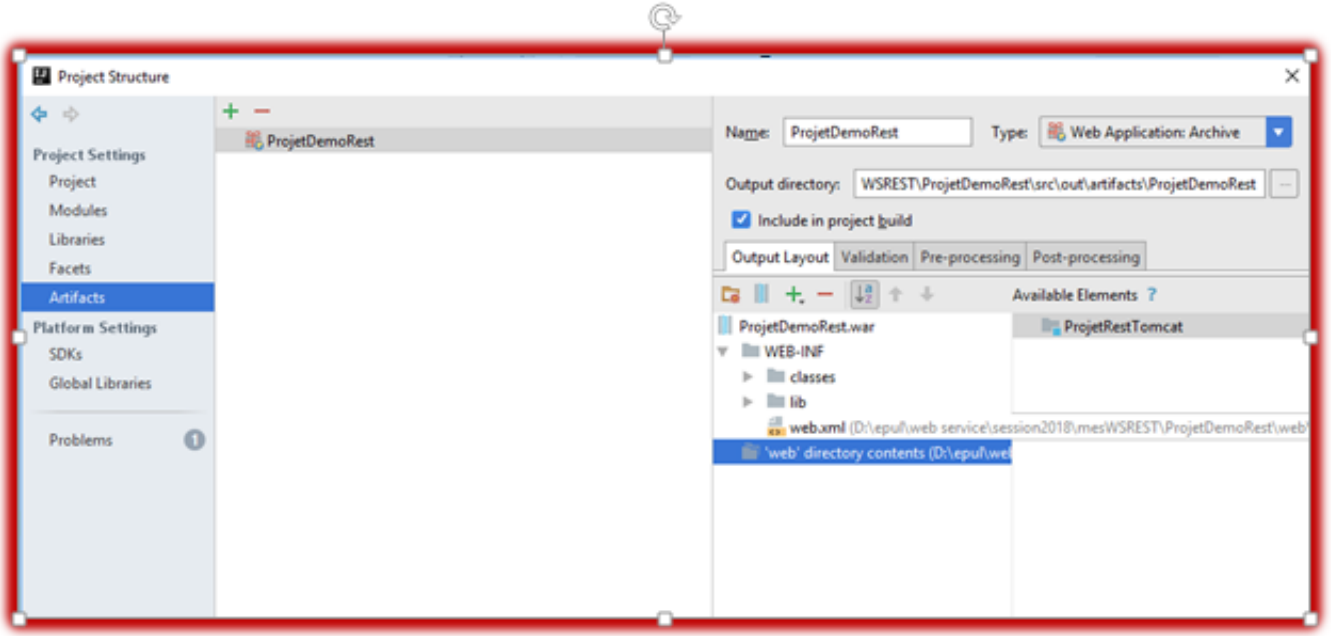
- **Serveur Tomcat**



WEB SERVICE

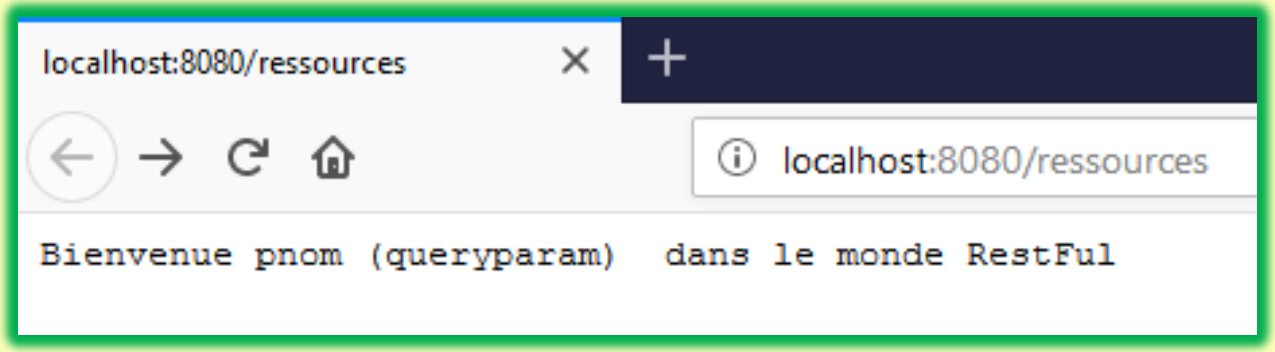
- **Artifacts**

Cette partie peut poser quelques difficultés car il faut produire un fichier.war avec les classes qui produisent les ressources.
Pour réaliser cette opération, je suis passé par un artifact exloded puis transformé en web archive.



WEB SERVICE

- **Exécution**

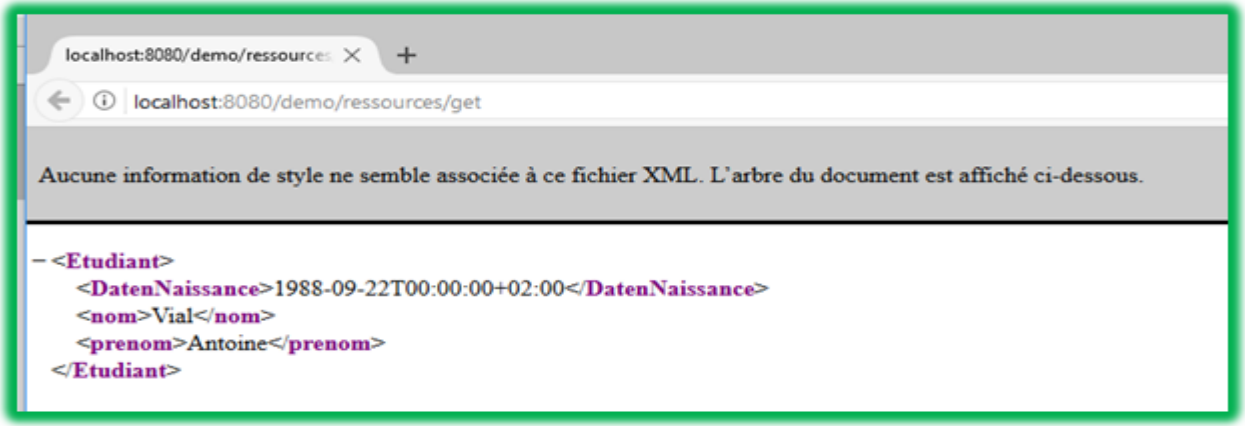


WEB SERVICE

- **Web Service RestFul**

-Test des fonctions

Sortie XML



Sortie JSON

Ecriture du Client

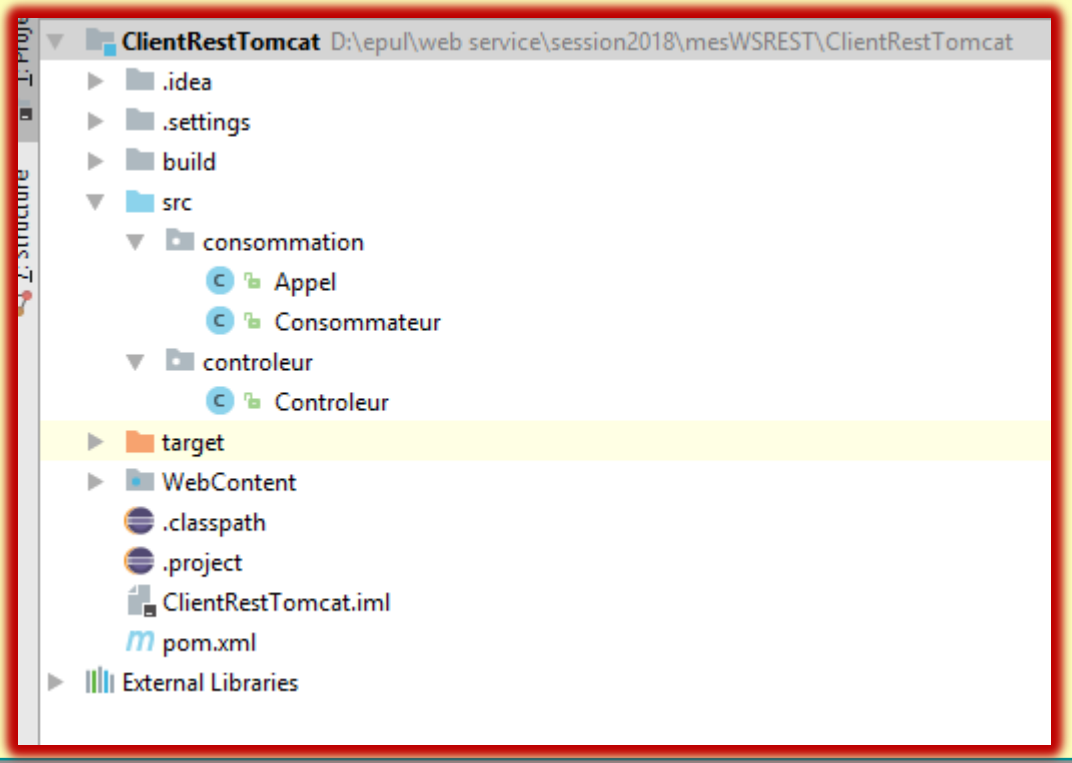
Consommateur

WEB SERVICE

- **Environnement de développement**

- Project Web -> Maven Projet

- Librairies : Tomcat 9. 0 Runtime



WEB SERVICE

- **Client RestFul du Web Service**

- Classe Consommateur

```
package consommation;


import ...

public class Consommateur {

    /**
     * @param args
     */

    protected static final String SERVICE_URI = "http://localhost:8080/demo/ressources";
    protected static Consommateur singleton = null;
    protected  Client client;
    protected  WebTarget target; // permet de récupérer l'URL du WS

    protected Consommateur() {

        client = ClientBuilder.newClient();
         target = client.target(SERVICE_URI);
    }

    protected static Consommateur get() {
        if (singleton == null)
            singleton = new Consommateur();
        return singleton;
    }

    protected WebTarget target() { return client.target(SERVICE_URI); }


}
```

WEB SERVICE

- **Client RestFul du Web Service**

- Classe Appel

```
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;

public class Appel {

    public  String  appelJson()
    {
        String uneChaine;

        WebTarget target = Consommateur.get().target;
        target = target.path("getjson");
        //System.out.println(" uri :"  + target.getUri());
        uneChaine= target.request().accept(MediaType.APPLICATION_JSON).get(String.class);
        return uneChaine;
    }
```

WEB SERVICE

- **Requête POST**

- Ajout d'un Adhérent (Nom, Prénom, Ville)

```
@POST
@Path("/Adherents/ajoutPost")
@Consumes("application/x-www-form-urlencoded")
@Produces("application/json")

public void ajouteAdherent( @FormParam("unAdh") String  unAdherent )
        throws MonException {

    DialogueBd unDialogueBd = DialogueBd.getInstance();
    Gson gson = new Gson();
    Adherent unAdh = gson.fromJson(unAdherent, Adherent.class);
    try {
        String mysql = "";
        mysql = "INSERT INTO adherent (nom_adherent, prenom_adherent, ville_adherent) ";
        mysql += " VALUES ( \'" + unAdh.getNomAdherent()+ "\', \'" + unAdh.getPrenomAdherent();
        mysql+="  \', \'"  + unAdh.getVilleAdherent() +  "\') ";
        unDialogueBd.insertionBD(mysql);

    } catch (MonException e) {
        throw e;
    }
}
```

WEB SERVICE

Requête POST

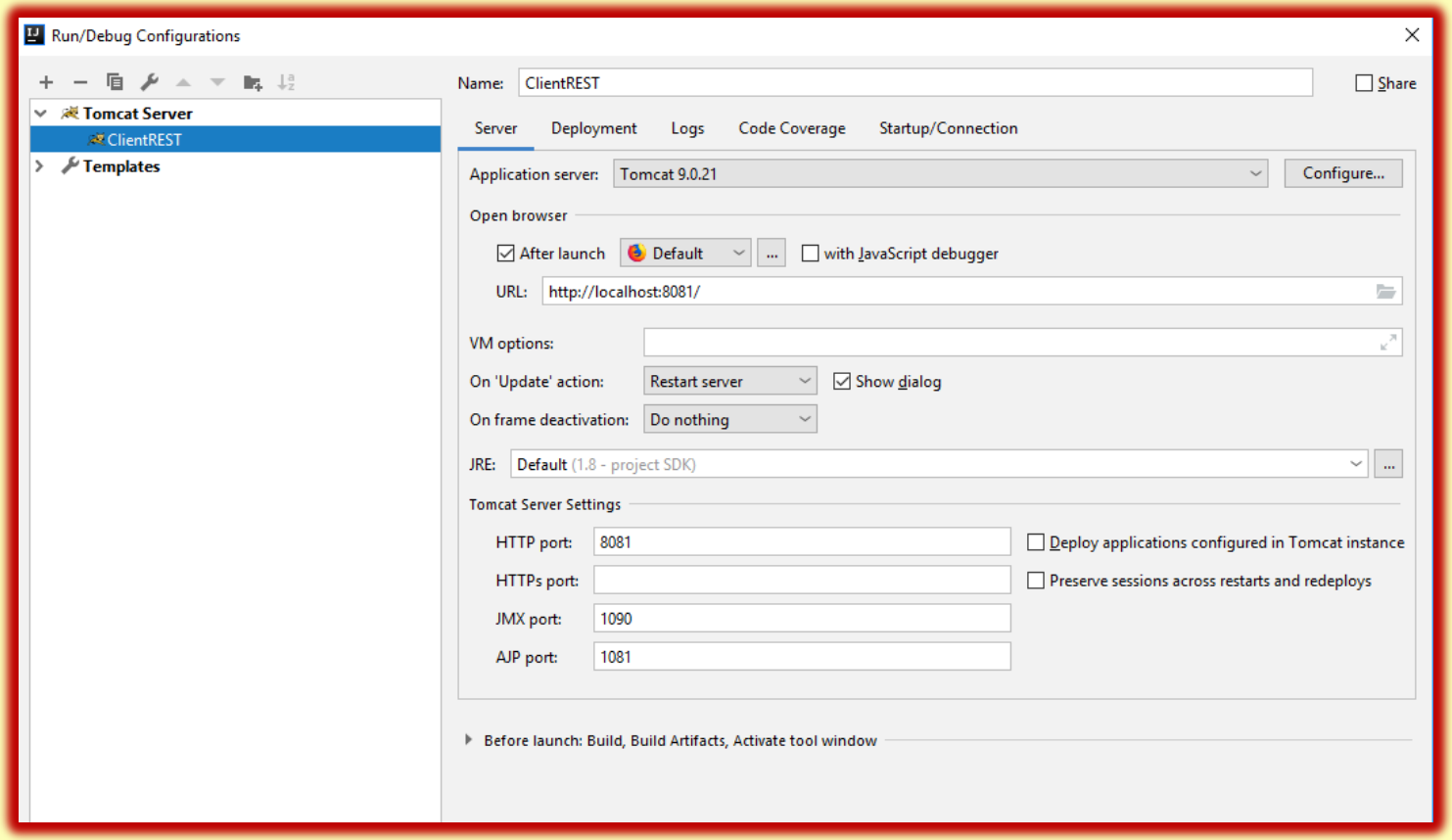
- Appel Client

```
public String postAjoutJson(String action, Object unObj)
{
    Response uneChaine;
    WebTarget target = Consommateur.get().target;
    target = target.path(action);
    System.out.println(" uri :"  + target.getUri());

    Form f = new Form();
    Gson gson = new Gson();
    String json=gson.toJson(unObj);
    f.param("unAdh", json);
    uneChaine = target.request().accept(MediaType.APPLICATION_JSON).
            post(Entity.entity(f, MediaType.APPLICATION_FORM_URLENCODED),Response.class);

    System.out.println(" passé");
    //Response response = target.request(MediaType.APPLICATION_JSON)
      //         .post(Entity.entity(json, MediaType.APPLICATION_JSON),Response.class);

     if(uneChaine.getStatus() >= 200 && uneChaine.getStatus() <= 299) {
         return "Mise Ã  jour effectuÃ©e";
     }
     else {
         return "Echec mise Ã  jour";
     }
}
```

On passe par un formulaire
import javax.ws.rs.core.Form;

WEB SERVICE

- **Configuration**



WEB SERVICE

- **Exécution**