# WEB SERVICE

# RESTFul

## REST

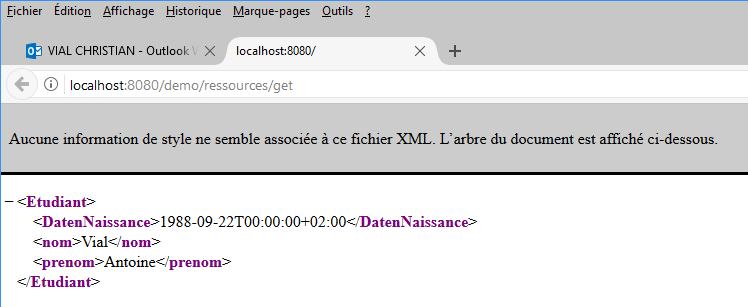
L'architecture REST n'est pas un protocole en soi, ni une technologie, mais une "philosophie" de l'utilisation du Web. Le protocole utilisé ici est simplement HTTP avec ses méthodes (GET, POST, DELETE et PUT…). Cette philosophie estime qu'il n'est, dans bien des cas, pas nécessaire de faire appel aux couches d'abstraction proposées par SOAP et XML-RPC, et que les méthodes de HTTP, combinées avec de bonnes URIs, suffisent amplement dans la majorité des cas.

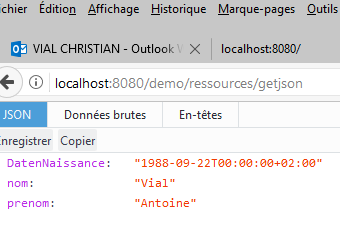
L’information de base, dans une architecture REST, est appelée **ressource**. Toute information qui peut être nommée est une ressource : un article d’un journal, une photo, un service ou n’importe quel concept. Dans un système hypermedia, une ressource est tout ce qui peut être référencé par un lien.

Une ressource est identifiée par un **identificateur de ressource**. Il permet aux composants de l’architecture d’identifier les ressources qu’ils manipulent. Sur le web ces identificateurs sont les URI (Uniform Resource Identifier).

### Voici un exemple de requête de type REST

Exemple





Ainsi lorsqu’on veut publier l’interface de son application sur le web, le client peut appeler les méthodes métiers à distance. On pourrait donc penser développer un service web REST en définissant un URI pour chacune de ces méthodes :

[**http://localhost:8080/demo/ressources/getjson**](http://localhost:8080/demo/ressources/getjson)

Mais si les paramètres sont plus complexes, on peut les mettre dans le corps de la requête. Mais là, des problèmes se posent et cette méthode devient vite très complexe voire incompréhensible (mon humble avis).

## Méthode REST ( Representational State Transfer)

### 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. Ceci est basé sur le chapitre 5 de la thèse de doctorat de Roy Fielding.

Dans l'architecture REST, des données et la fonctionnalité sont considérés comme des ressources, et ces ressources sont accessibles par l'URI. Le principe de base est «ressources-oriented architecture".

RESTful Web Services sont les services Web qui sont construites sur le architecture REST. Ces services exposent les ressources sur le Web URIs. Les méthodes HTTP suivantes sont utilisés pour créer, récupérer, mettre à jour et effacer des ressources:

|  |  |
| --- | --- |
| * **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. |

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)
* HTML (text/html)
* JavaScript Object Notation (application/json)
* XML (application/xml)

@ ProduceMime ( "text / plain") - spécifie le format de sortie le plus simple, les autres formats sont par exemple: XML, JSON et HTML

\* @Path("nom") donne le nom d’un espace pour accéder aux méthodes de la classe

***@Path("bonjour")***

***public class WsSalutation {……***

\* @ Path ( "/ Etat / État () / {params :.\*}") - (params: .\*) est un espace qui représente un certain nombre de paramètres dynamiques

@ PathParam ( "name") permet de passer des paramètres par le segment PathSegmen

@GET

@Path("test/{Id}")// récupère la valeur passéé par :

webResource.path("hello/test").path("brun")

@Produces("text/plain")

public String donneunautrebonjour(@PathParam("Id") String id)

#### 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](http://en.wikipedia.org/wiki/Web_Application_Description_Language) décrit le web service. | [WSDL](http://en.wikipedia.org/wiki/Web_Services_Description_Language) décrit le web service. |
| Léger facile à developer . | Lourd, complexe à développer. |
| Lié au protocole HTTP | Incorporé dans l’appel RPC |

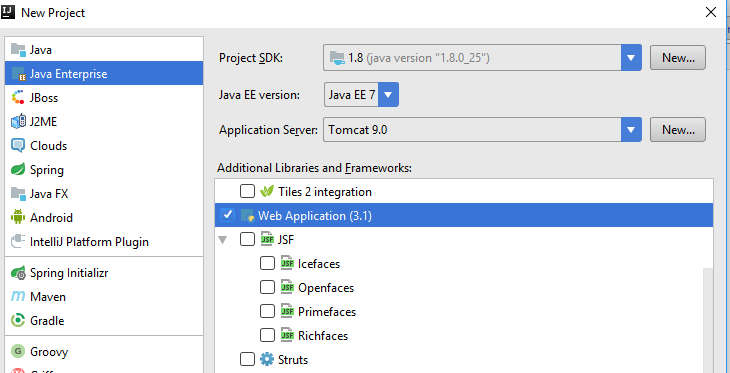
### Conclusion

Il est facile, par les trois exemples proposés par un même service Web, de voir que la méthode la plus simple est REST, qui a l'avantage énorme de ne pas ajouter une couche d'abstraction à des données qui n'en ont pas forcément besoin.

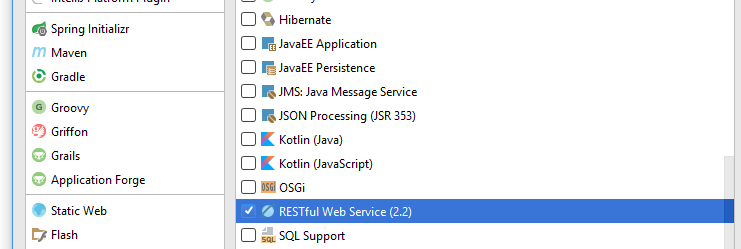
# Construction d’un Web Service RestFul

## ProjetRestTomcat

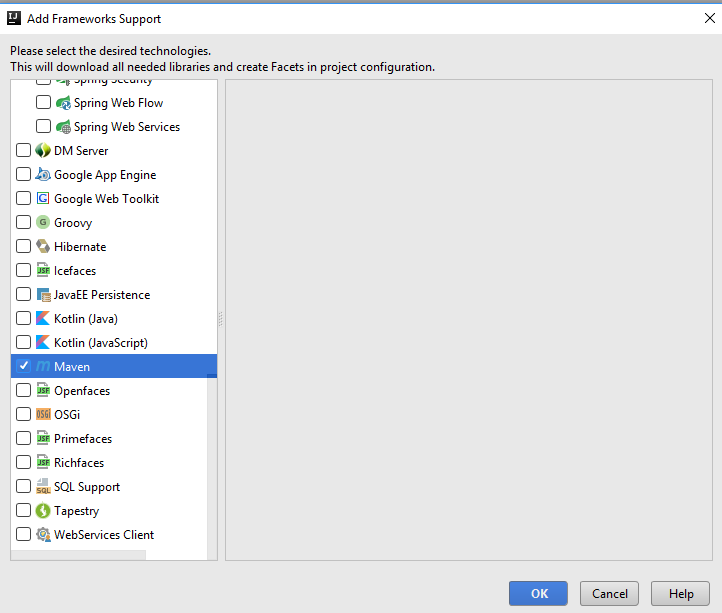
Démarrez un nouveau projet nommé ProjetRestTomcat



Avec l’option

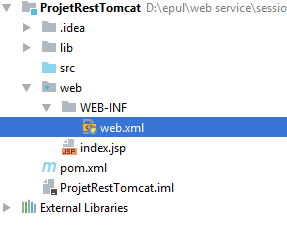


Ajouter l’option maven



Supprimez les packages main et testJava

Voici l’architecture de votre projet



### Maven : ajout des dépendances

Vous devez ajouter les dépendances des bibliothèques jersey pour produire un web service RestFul

<**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**>  
 <**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**>

### Ajout des packages

On crée les packages metier et ws en ajoutant les classes suivantes :

#### Classe Etudiant

**package** metier;  
**import** javax.xml.bind.annotation.XmlElement;  
**import** javax.xml.bind.annotation.XmlRootElement;  
**import** java.util.\*;  
  
@XmlRootElement(name = **"Etudiant"**)  
**public class** Etudiant {  
   
 **private** String **nom**;  
 **private** String **prenom**;  
 **private** Date **dnaissance**;  
  
 **public** Etudiant() {  
 }  
**public** Etudiant(String nom, String prenom, Date dnaissance) {  
 **super**();  
 **this**.**nom** = nom;  
 **this**.**prenom** = prenom;  
 **this**.**dnaissance** = dnaissance;  
}  
  
**public void** setDnaissance(Date dnaissance) {  
 **this**.**dnaissance** = dnaissance;  
}  
  
**public void** setNom(String nom) {  
 **this**.**nom** = nom;  
}  
  
**public void** setPrenom(String prenom) {  
 **this**.**prenom** = prenom;  
}  
  
@XmlElement  
**public** String getNom() {  
**return nom**;  
}  
@XmlElement  
**public** String getPrenom() {  
 **return prenom**;  
}  
@XmlElement(name = **"DatenNaissance"**)  
**public** Date getDnaissance() {  
 **return dnaissance**;  
}  
  
}

#### Classe RestFulApplication

Cette classe est indispensable pour lancer le Web Service.

**package** ws;  
  
**import** java.util.HashSet;  
**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**;  
 }  
}

#### 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() {  
 }

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/ProjetWebRestFull/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!"**;  
 }

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/ *ProjetWebRestFull* /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;  
 }

Ressource XML

On produit des informations au format json

@GET  
 @Path(**"/getjson"**)  
 @Produces(**"application/json"**)  
 *// http://localhost:8080/ ProjetWebRestFull /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;  
  
 }

*//  
 // Appel du paramètre avec la notation ?  
 // /* @GET  
 @Produces(**"text/plain"**)  
 *// http://localhost:8080/ ProjetWebRestFull /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"**;  
 }

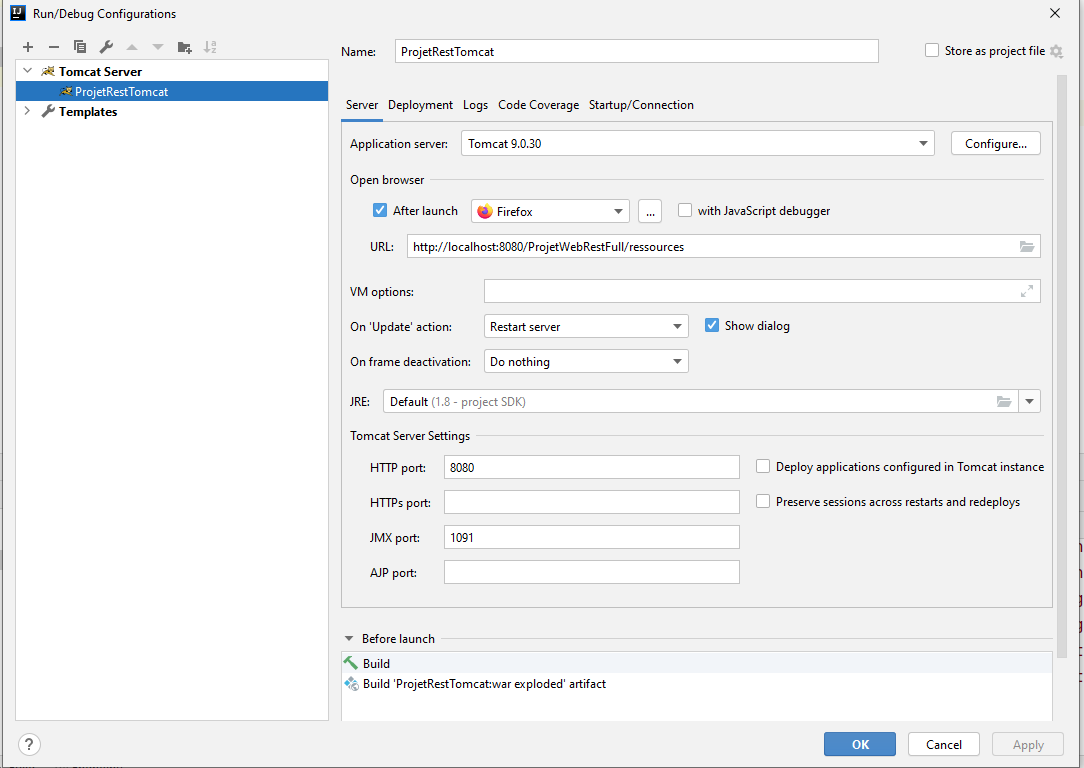
*// /  
 // Utilisation de la méthode Post  
 // //* @POST  
 @Produces(**"text/plain"**)  
 @Consumes(**"multipart/form-data"**)  
 **public** String salutationMethodePost(String pnom) {  
 **if** (pnom != **null**) {  
 **return "Bonjour pnompost (QueryParam) "** + pnom  
 + **" et bienvenue dans le monde Restfull!"**;  
 }  
 **return "Bienvenue pnompost (queryparam) dans le monde RestFul"**;  
 }

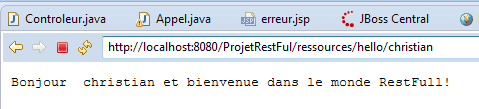
*//  
 // Format de sortie : Html* @GET  
 @Path(**"html/{Id}"**)  
 *// récupère la valeur passéé par webResource.path("html").path("xxxx")* @Produces(**"text/html"**)  
 *// http://localhost:8080/ ProjetWebRestFull /ressources/html/christian* **public** String SalutationHtml(@PathParam(**"Id"**) String 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!"**;  
 }  
}

Test du Web Service

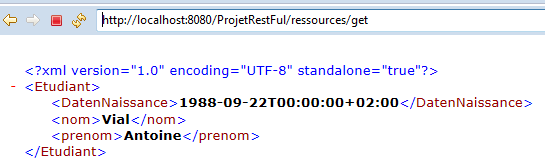
### Déploiement du web service au format war sous Tomcat

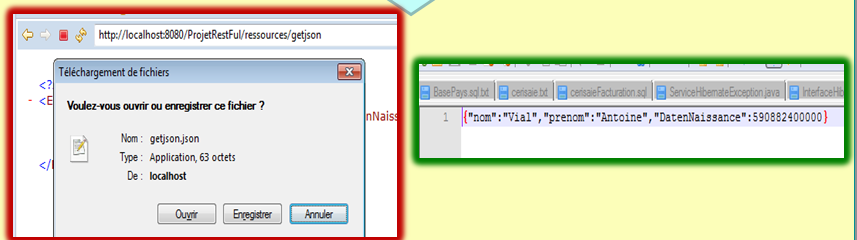
Vous pouvez tester le web service sous un navigateur en entrant les URI présentés dans le cours.







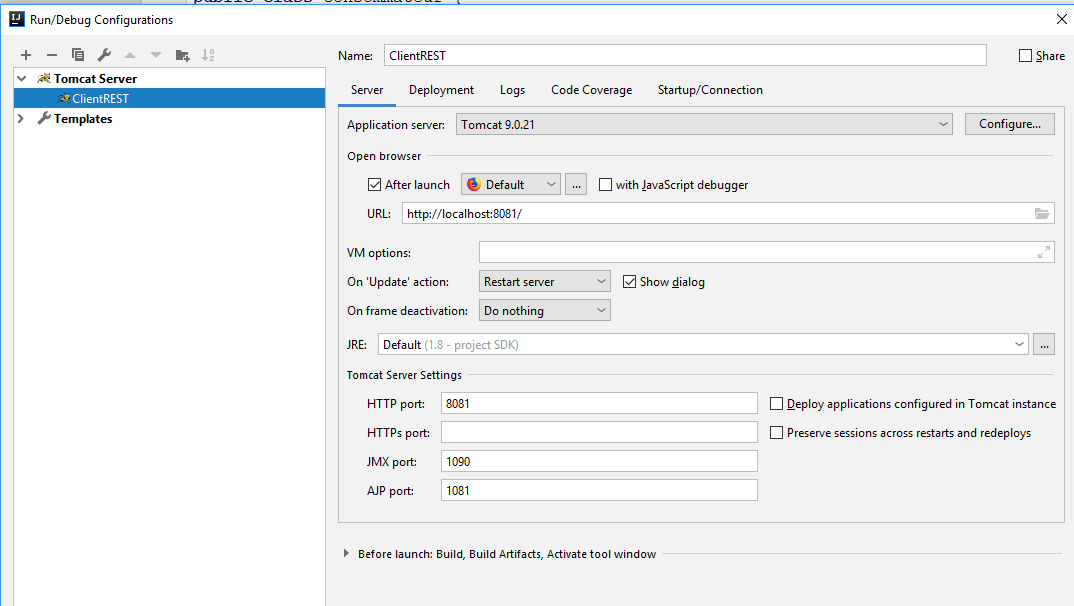




## Client consommateur

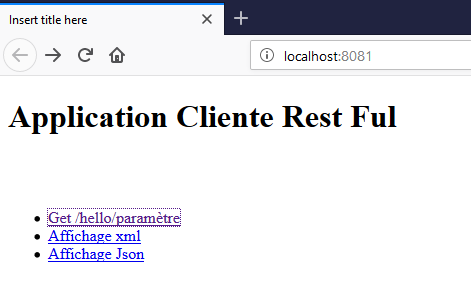
Déployez le client consommateur sous Tomcat et testez ces appels.

Vous pouvez noter la configuration qui va utiliser un autre port sous Tomcat



## Travail à faire

Testez les méthodes présentes :



**Vous pouvez ajouter :**

* une méthode qui réalise une conversion d’une température exprimée en degré pour la passer en fahrenheit
  + public double donneCelciustoFahrenheit(double t) {
  + double fahrenheit;
  + fahrenheit = 9.0 / 5.0 \* t + 32;
  + return fahrenheit;
  + }
  + public double donneFahrenheittoCelcius(double t) {
  + double celcius;
  + celcius = (t - 32) \* 5.0 / 9.0;
  + return celcius;
  + }

Le client appellera le webservice en passant les paramètres en mode :

* Path
* Query