



# TP2 : Les Services Web avec JAX-RS

**3BI1/3BI2/3WEB/3RSA/3SEM**

**Objectifs du TP :** *Création de services web de type REST avec JAX-RS, l'IDE Netbeans et le serveur Glassfish*

## I. Les Services Web de type REST

### I.1. Présentation de REST

**REST** (**R**epresentational **S**tate **T**ransfer) est un style d'architecture basé sur les standards Web et le protocole http. Il a été présenté en premier lieu par Roy Fielding en 2000.

Dans une architecture basée sur REST :

- Tout est **Ressource**. Une ressource est accédée via une interface basée sur les méthodes http standard.
- Il existe un serveur REST qui fournit l'accès aux ressources, et un client REST qui accède et modifie les ressources REST.
- Chaque ressource doit supporter les opérations HTTP communes. Elle est identifiée par un ID global (typiquement un URI)
- Les ressources peuvent avoir des représentations différentes (texte, xml, json...). Le client REST peut exiger des représentations spécifiques via le protocole HTTP (*négociation de contenu*).

### I.2. Les Méthodes http

Les méthodes *PUT*, *GET*, *POST* et *DELETE* sont utilisées dans les architectures à base de REST.

- **GET** définit un accès en lecture de la ressource. Il ne change jamais la ressource
- **PUT** crée une nouvelle ressource
- **DELETE** supprime une ressource
- **POST** met à jour une ressource existante et peut en créer des nouvelles

### I.3. Les Services Web RESTful

Les services web RESTful sont basés sur les méthodes http et le concept de REST, Ils définissent un URI de base pour les services, les types MIME (XML, Text, JSON, ...) et l'ensemble des opérations supportées (POST, GET, PUT et DELETE).

## II. Présentation de JAX-RS

Java définit un support pour le standard REST via JAX-RS (*Java API for RESTful Web Services*). Il utilise des annotations pour simplifier le développement de services web REST. Ces annotations permettent de définir les ressources et les actions pouvant y être appliquées. Ces annotations prennent effet à l'exécution, pour générer les classes et artefacts pour la ressource.

Le tableau ci-dessous représente les annotations les plus utilisées dans JAX-RS.

Annotation	Description
<b>@PATH</b>	Chemin URI indiquant où une classe Java sera hébergée. Par exemple <i>/helloworld</i> . Il est également possible d'insérer des variables dans un URI, par exemple, il est possible de demander le nom de l'utilisateur et le passer à l'application dans le chemin : <i>/helloworld/{username}</i>
<b>@GET</b>	Une méthode Java annotée avec @GET désigne une requête de type HTTP GET. Le comportement de la ressource est déterminé par la méthode HTTP à laquelle la ressource répond.
<b>@POST</b>	Désigne une méthode correspondant à la requête HTTP POST.
<b>@PUT</b>	Désigne une méthode correspondant à la requête HTTP PUT.
<b>@DELETE</b>	Désigne une méthode correspondant à la requête HTTP DELETE.
<b>@HEAD</b>	Désigne une méthode correspondant à la requête HTTP HEAD.
<b>@PathParam</b>	Désigne un paramètre pouvant être extrait à partir de l'URI, pour être utilisé par la ressource. Le nom du paramètre entré correspond au nom de la variable définie dans l'annotation @PATH
<b>@Consumes</b>	Utilisée pour spécifier le type de représentation MIME qu'une ressource peut consommer, et qui sont envoyées par le client.
<b>@Produces</b>	Utilisée pour spécifier le type de représentation MIME qu'une ressource peut produire, et envoyer au client.

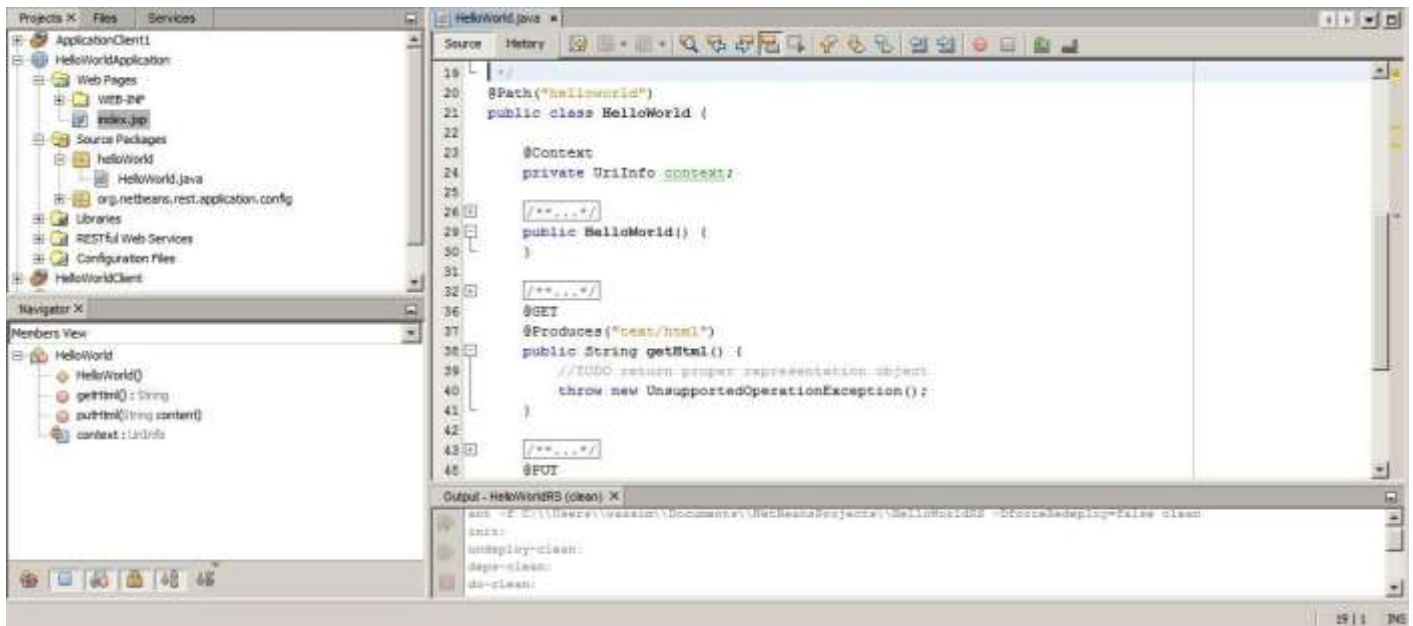
### III. Première Application JAX-RS : Helloworld

#### III.1. Création du service web

Pour créer un service Web RESTful en utilisant l'IDE NetBeans et Glassfish (installés et configurés dans le TP précédent), procéder comme suit :

1. Dans NetBeans, créer une application web simple, que vous nommerez HelloWorldApplication
2. Faire un clic-droit sur le projet, et sélectionner *New -> RESTful Web Services from Patterns*.
  - a. Désigner comme design pattern *Simple Root Resource*
  - b. Utiliser comme nom de package : *helloWorld*
  - c. Utiliser comme Path : *helloworld*
  - d. Utiliser comme nom de classe : HelloWorld

e. Utiliser type MIME *text/html*  
Votre Projet aura ainsi l'apparence suivante :



3. Dans le fichier *HelloWorld.java*, la méthode *getHTML()* représente une requête de type GET. Remplacer le commentaire *//TODO* par le code suivant, pour afficher votre message :

```
return "<html LANG=\"en\"><body><h1>Hello, World!!</body></h1></html>";
```

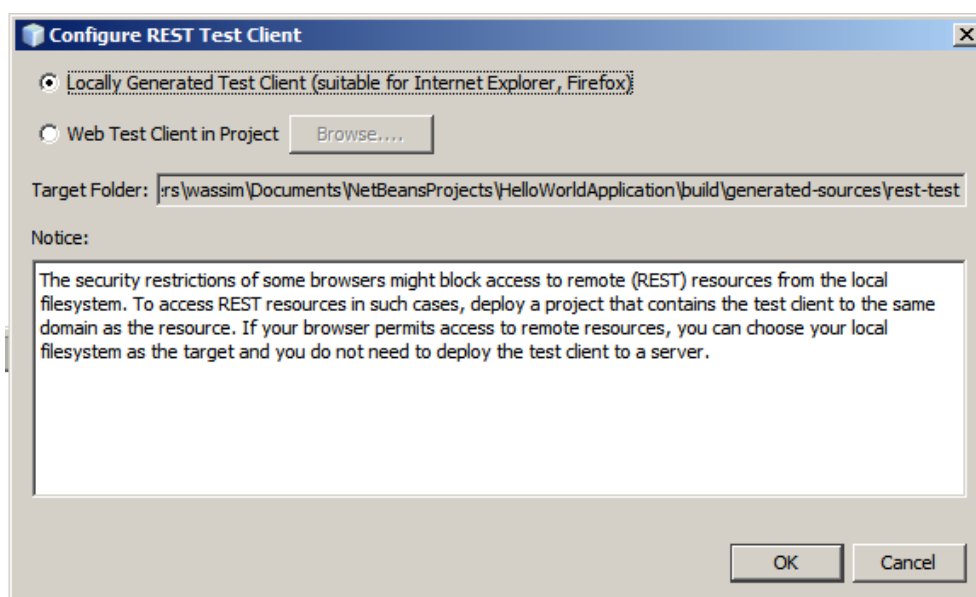
Le code obtenu sera alors le suivant :



### III.2. Test du service web

Pour tester le service web, suivre les étapes suivantes :


1. Clic-droit sur le projet et choisir *Test Restful Web Services*. Cela ouvrira la fenêtre suivante :



2. Choisir *Web Test Client in Project*, et sélectionner le projet *HelloWorldApplication* dans la fenêtre qui vous est offerte.
3. Un navigateur web va s'ouvrir, avec l'interface de test.
  - a. Choisir votre ressource *helloworld* dans le panneau gauche de votre écran
  - b. Choisir la méthode à tester (dans notre cas, c'est la méthode GET)
  - c. Cliquer sur le bouton *Test*.L'affichage suivant doit apparaître dans votre écran :

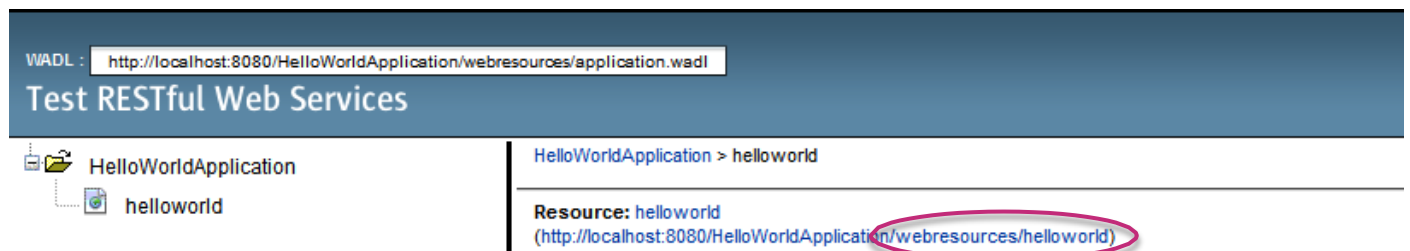


### III.3. Configuration du Run

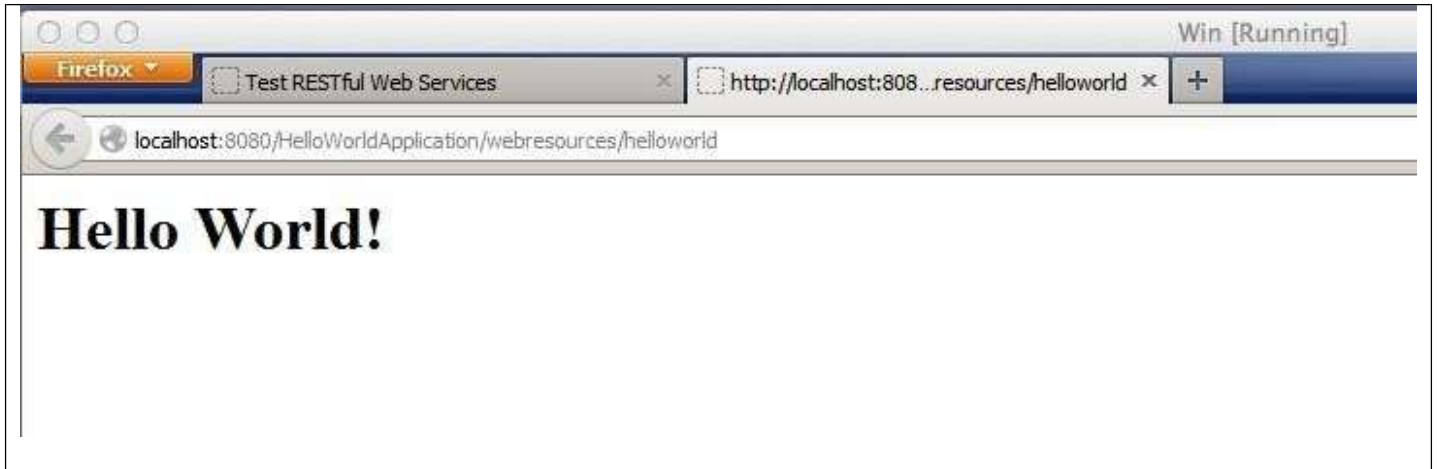
Pour configurer l'exécution de votre application avec le bouton *Run* , suivre les étapes suivantes :

1. Clic-droit sur le nom du projet et choisir *Properties*
2. Sélectionner la catégorie *Run* dans la fenêtre qui apparaît
3. Mettre la valeur du champs *Relative URL* à l'emplacement du service web REST relatif au PATH, qui est, pour cet exemple : *webresources/helloworld*

**Remarque :** pour déterminer la valeur du *Relative URL* dans l'exemple de test que vous avez réalisé dans la partie III.2. , regarder le chemin affiché en haut de votre navigateur, dans le panneau de droite :



4. Déployer votre service web
5. Exécuter votre service web. L'affichage suivant apparaît dans le navigateur.



## IV. Services Web RESTful à partir d'une base de données

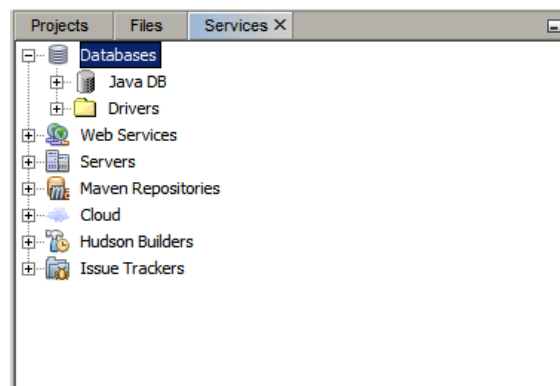
### IV.1. Création et configuration de la base de données

On se propose de générer un service web RESTful à partir d'une base de données de notre choix. Pour ce faire, commencer par créer la base de données avec MySQL (en utilisant l'outil d'administration de votre choix). Appeler la base « maBD » et créer une table *client* dans cette base.

Choisir une structure et insérer un contenu dans cette table.

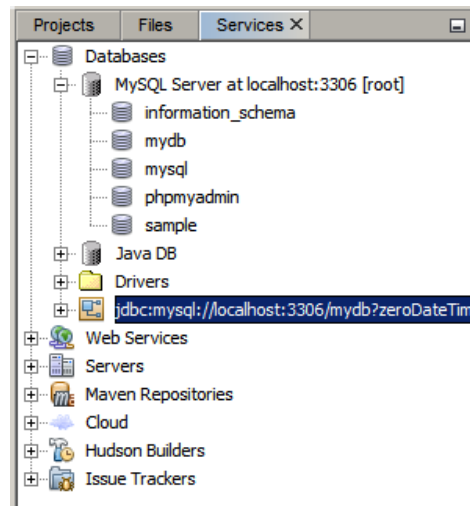
Une fois la base de données créée, il faut la configurer au niveau de NetBeans. Pour cela, suivre les étapes suivantes :

- Aller sous l'onglet *Services* dans le panneau gauche de votre IDE



- Clic-droit sur *Databases* et choisir *Register MySQL Server*
- Garder les paramètres par défaut, et valider.

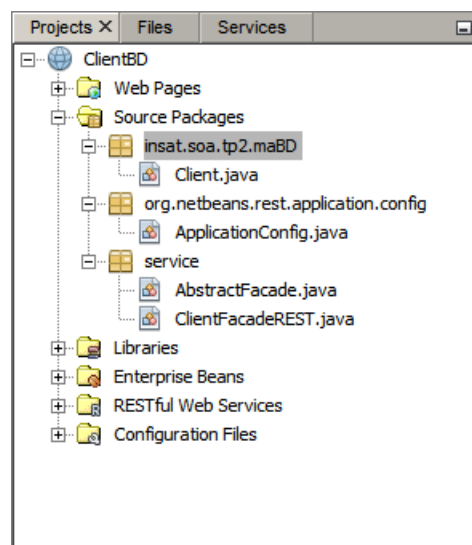
- Clic-droit sur le nouveau serveur MySQL qui est apparu, et choisir *Connect*. La liste des bases de données disponibles dans le serveur apparaît, dont la base *maBD*.
- Clic-droit sur *maBD* et choisir *Connect*. Une connexion à la base de données sera alors créée, comme suit :



## IV.2. Génération du service web

Pour générer le service web qui utilise cette base de données, suivre les étapes suivantes :

- Créer un nouveau projet de type Application Web, qu'on nommera *ClientBD*
- Clic-droit sur ce projet, et faire *New -> Restful Web Services from Database*
- Créer une nouvelle source de données relative à votre base. Si tout se passe correctement, la liste des tables dans la base apparaît dans le champs *Available Tables* à gauche de la fenêtre.
- Ajouter la table *client* à la liste des *Selected Tables*
- Choisir comme nom de package *itbs.soa.tp2.maBD*
- Valider. Votre projet aura alors l'arborescence suivante :





Regarder le code généré, et essayer de comprendre les différentes fonctionnalités. Lancer un Test du service web, et essayer d'exécuter l'ensemble des méthodes HTTP existantes. Noter la différence entre elles.

## V. Services Web RESTful à partir d'objets Java

On désire dans ce qui suit créer un service web RESTful dont les ressources sont représentées par des classes Java annotées.

Le Service Web REST de cet exercice consiste à créer un système CRUD pour l'interrogation et la réservation de trains. Les ressources manipulées par les services sont donc un **train** et une **réservation**. Le service Web REST doit pouvoir lister l'ensemble des trains, et de lister les trains qui satisfont un critère de recherche (nom, ville de départ, ville d'arrivée).

Pour cela, suivre les étapes suivantes :

1. Créer une application web, appelée *ReservationTrains*
2. Créer la classe *Train*, qui modélise le concept de trains, et qui contient un attribut *String nom*, et deux attributs *String villeDepart* et *String villeArrivee* représentant les villes de départ et d'arrivée du train. Ajouter les getters et les setters de cette classe. Le code ressemble à ce qui suit (**c'est un code manquant, vous devez le terminer**):

```

package insat.soa.tp2.train;

import javax.xml.bind.annotation.XmlRootElement;

/**...*/
@XmlRootElement(name="train")
public class Train {

    private String nom;
    private String villeDepart;
    private String villeArrivee;

    public Train() {
    }

    public Train(String name, String villeDepart, String villeArrivee) {
        this.nom = name;
        this.villeDepart = villeDepart;
        this.villeArrivee = villeArrivee;
    }

    public String getNom() {
        return nom;
    }
}

```

Annotation qui permet de convertir cet objet Java en XML et vice-versa.

3. Créer la classe *ReserverTrain* qui permet de persister les informations concernant le service web.

```

public class ReserverTrain {

    private static List<Train> trains = new ArrayList<Train>();

    static{
        trains.add(new Train("TR1", "Tunis", "Sousse"));
        trains.add(new Train("TR2", "Sfax", "Gabes"));
    }

    public static List<Train> getTrains() {
        return trains;
    }

}

```

Modifier ces données à votre guise

4. Créer la classe *TrainRessource* qui permet l'accès aux services web RESTful de la ressource *Train*. On se propose de réaliser les fonctions suivantes (cette classe est à implémenter vous-mêmes) :

- Définir comme chemin de ressource racine la valeur `/train`
- Créer la méthode `getTrains`, qui permet d'afficher la liste des trains sous le format JSON.
- Créer la méthode `getTrain`, qui utilise un chemin de ressource racine `/train/nom/{nom}`, et qui permet d'afficher le train dont le nom est donné comme paramètre à la requête. Le résultat de cette méthode sera sous la forme XML.

**Indication :** Compléter pour cela le code suivant :

```

...
...
@Path("nom/{nom}")
public Train getTrain(@PathParam("nom") String nomTrain) {
    ...
}

```

Compléter la méthode HTTP nécessaire

Compléter pour définir le format de retour

Ceci est le path à ajouter au chemin principal du service web. Il permet de saisir comme paramètre une valeur entrée par l'utilisateur, représentée ici par {nom}

Ceci indique que le paramètre `nomTrain` représente le même paramètre entré par l'utilisateur dans le chemin d'accès : `nom`

- Créer la méthode `rechercheTrain`, qui utilise un chemin de ressource racine `/train/recherche`, et qui permet, selon deux critères `villeDepart` et `villeArrivee` entrés en paramètre, d'afficher la liste des trains correspondant à ces deux critères. Le résultat de cette méthode sera sous la forme XML.

**Indication :** Compléter pour cela le code suivant :

```

...
...
...
public List<Train> rechercheTrain(@QueryParam("depart") String dep,
    @QueryParam("arrivee") String arr) {
    ...
}

```

Compléter la méthode HTTP nécessaire

Compléter pour définir le chemin demandé

Compléter pour définir le format de retour demandé

Ceci indique que les paramètres `dep` et `arr` correspondent aux paramètres `depart` et `arrivee` entrés par l'utilisateur dans la requête.

5. A partir du navigateur, invoquer les trois services comme suit :

- <http://localhost:8080/Train/webresources/train>
- <http://localhost:8080/Train/webresources/train/nom/TR1>
- <http://localhost:8080/Train/webresources/train/rechercheTrain?depart=Tunis&arrivee=Sousse>

## VI. Homework

Reprendre l'exemple de convertisseur de monnaie développé pour le TP précédent, et l'appliquer sur les services web REST. Le but est d'utiliser les différentes méthodes HTTP (`@Get`, `@Post`, `@Put` et `@Delete`), ainsi que la saisie de paramètre dans le path (`@PathParam`) et dans la requête (`@QueryParam`).

*Date de remise du travail : prochaine séance de TP.*