TP2

Services Web REST Personnalisé avec JAX-RS

Objectif:

- Créer et personnaliser des **web services** de type REST en Java en utilisant la classe Response de la bibliothèque Jax-rs

Outils nécessaires :

❶ **Eclipse IDE for Enterprise Java and Web Developers**

Lien de téléchargement : Eclipse downloads - Select a mirror | The Eclipse Foundation

❷ **WildFly (**dans nos TP, nous utilisons la version widldfly 29.0.1 Final**)**

Lien de téléchargement : WildFly

❸ **Postman** logiciel de test des web sevices

Lien de téléchargement : Download Postman | Get Started for Free

Activité : Personnaliser les web services avec l'objet Response

Quelle que soit l'opération demandée, les ressources indiquent toujours un **statut de retour**. Ce code est renvoyé au client afin que celui-ci puisse connaître le problème réel et procéder en conséquence. 200, 500 et 404 sont les plus connus, ils indiquent, dans l'ordre, un succès, un échec avec une erreur serveur et l'absence du document recherché. Nous allons utiliser un générateur **Response** pour générer une réponse avec le code souhaité.

1. Créez un nouveau projet web Dynamique « tp2_ex1 ».
2. Y-ajoutez un package « **com.isitic.rest** » contenant 3 classes **Etudiant**, **EtudiantRest** et **RestActivator**.
3. Créez la classe étudiant qui est identifiée par un id (int) et ayant un nom et une moyenne. Ajoutez un constructeur sans paramètres, un constructeur avec 3 arguments qui initialise tous les attributs, les getters et les setters.

4. Créez la classe **EtudiantRest**, y-ajoutez un attribut **private static ArrayList<Etudiant>liste** , et un constructeur initialisant cette liste par 3 étudiants.

```
@Path("/etudiants")
	public class EtudiantRest {
	private static ArrayList<Etudiant> liste;

	public EtudiantRest() {
	  liste = new ArrayList<Etudiant>();
	  liste.add(new Etudiant(1, "Ahmed", 12.5));
	  liste.add(new Etudiant(2, "Rihab", 14));
	  liste.add(new Etudiant(3, "Omar", 15));
	}
}
```

5. Ajoutons maintenant à cette classe une fonction ayant les caractéristiques suivantes :

Fonction	URL	Méthode http	Suucès/Echec	Code Statut
getAll() : retourne la liste de tous les étudiants en format json	RestApi/etudiants	GET	Succès	200 ok
			No content (liste vide)	204

Voici le code de la fonction :

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response getAll(){
	if(liste.size()==0)
		return Response.status(204).build();
		//return Response.status(Status.NO_CONTENT).build();
		//return Response.noContent().build();
		//avec le test niveau postman, la liste ne sera pas affichée
		//même si l'objet Entity est initialisé
		//Rappel : le code 204 n'affiche pas l'objet Entity
		//return Response.status(204).entity(liste).build();
	else
		return Response.status(Status.OK).entity(liste).build();
}
```

Testez cette ressource.

6. Ajoutez maintenant les fonctions suivantes à la classe **EtudiantRest**:

Méthode	URL	Méthode Http	Succes/Failure	Code Statut
getEtudiant(int id) :	RestApi/etudiants/etudiant?	GET	Succès	200

Retourne un étudiant ayant l'id correspondant	id=valeurId		Inexistant	204
			Erreur serveur	500
deleteEtudiant(int id) : supprime un etudiant selon l'id. Si l'id est inexistant, alors afficher un message	RestApi/etudiants/{id}	DELETE	Succès	200
			Inexistant	224
			Erreur serveur	500
updateEtudiant(int id, String nom, double moyenne) : mise à jour des données de l'étudiant s'il existe sinon création d'un nouveau étudiant	RestApi/etudiants/{id}	PUT	Succès (mise à jour)	200
			Succès (création)	201
			Erreur serveur	500
createEtudiant(Etudiant std) : ajouter l'étudiant à la liste s'il n'existe pas sinon afficher un message	RestApi/etudiants/	POST	Succès	201
			Succès (id existant)	200
			Erreur serveur	500

7. Modifiez la méthode **getEtudiant** afin d'afficher un message indiquant que l'étudiant n'existe pas.

Exercice :

1. Ajoutez une classe ***groupe*** décrite par un identifiant, un nom de groupe et une liste d'étudiants relative à ce groupe
2. Ajoutez une classe implémentant des web services de manipulation de ***groupes***. Il faut **varier** les services afin de couvrir :
   a. Toutes les méthodes HTTP
   b. Tous les types de paramètres : paramètres de chemin, de requête, de formulaire et JSON.
   c. Des codes HTTP prédéfinis et personnalisés