Aziliz Baron
Ismail Mellali

Compte rendu TP 2 et 3 : JPA et Servlet/REST

Ces deux TP ont pour objectif de nous faire découvrir l'interface de programmation JPA dans un premier temps. Puis la notion de servlet en la combinant avec JPA et les principes d'une architecture REST.

TP2 : JPA

1. Déclaration des entités

Nous avons commencé par transformer une classe en entité. Pour cela il fallait d'abord importer les packages correspondant : **javax.Persistence.**

Une fois importé, on a indiqué devant les classes `@Entity` pour déclarer la classe en entité. Il faut ensuite renseigner la clé primaire de l'entité et indiquer qu'elle sera générée automatiquement par le SGBD, pour cela on utilise les annotations : `@Id @GeneratedValue.`

2. Déclaration des associations entre entités

Le lien entre deux entités peut se faire avec trois annotations différentes :

- `@ManyToOne` : utilisé quand il s'agit d'une relation n : 1
- `@OneToMany` : utilisé quand il s'agit d'une relation 1 : n. Il faut lui rajouter : `(mappedBy="id")` qui permet de référencer le champ qui porte la relation côté maître.
- `@ManyToMany` : utilisé quand il s'agit d'une relation n : n

3. Utilisation de JPA

On a intégré une classe jpaTest.java qui utilise une *EntityManager.* Cette classe va peupler la base de données et y faire des requêtes.

Cette classe ajoute ainsi une personne nommée Martin. Puis par une autre requête affiche toutes les personnes insérées dans la base de données.

Nous avons aussi intégrer une relation d'héritage. Les classes *Chauffage* et *Equipement* vont maintenant hériter de la classe *PeriphIntelligent.* Grâce à l'ajout de `@Inheritance(strategy = InheritanceType.SINGLE_TABLE)` devant la classe *PeriphIntelligent* ont indique qu'il y aura une seule table de créée comportant les classes *Chauffages* et *Equipement*

4. Mise en évidence du problème de n+1 select

Nous constatons qu'avec l'utilisation de Joinfetch, le temps de réponse est bien mieux que celui avec N1Select puisqu'une seule requête est effectuée avec Joinfetch.

TP3 : Servlet

Nous avons commencé par insérer une dépendance et un plugin dans le pom.xml pour pouvoir faire fonctionner le servlet.

Ensuite nous avons créé une classe *MyServlet* étendu à la classe *HttpServlet* et ajouter un fichier html appelé *MyForm* et une classe *UserInfo.* Tout cela nous a permis de tester le servlet en get et en post.

Ensuite l'objectif était d'insérer ce mécanisme au TP fait précédemment.

Pour cela nous avons, créé un formulaire *inscription* qui permet à une personne de s'inscrire. Puis une classe *PersonneInfo* qui contient la méthode doPost qui va permettre d'enregistrer dans la base de données les personnes s'inscrivant via le formulaire. Et la méthode doGet qui permet d'afficher les personnes inscrites dans la base de données.

Pour finir nous avons créé la couche de service pour notre application que l'on a nommé ResidenceRessource.