TP JEE (2)
Logic metier et Entreprise Java Beans

Les EJB (Enterprise JavaBeans) 3.0 permettent de découpler la logique de présentation (site web, application riche, services webs) de la logique métier (implémentation de services, interaction avec la base de données). En conséquence, la logique métier doit être exécutée dans un objet différent de celui qui gère l'affichage. De plus, ils autorisent la communication avec d'autres services répartis sur d'autres serveurs d'applications via des mécanismes de communication comme JNDI (Java Naming and Directory Interface).

FIGURE 1 – Architecture multi-niveaux : la couche de services

Nous verrons dans ce TP deux sortes d'EJB :
- **EJB session** : il s'agit d'objets qui rendent un service et qui permettent d'assurer une transaction. Il existe deux sortes d'EJB session : les EJB Stateless dont la particularité principale est de ne pas conserver d'état entre les différents appels, les EJB Statefull qui conservent un état entre différents appels de méthodes.
- **EJB Entity** : il s'agit d'objets qui correspondent à des enregistrements d'une base de données et qui gèrent l'accès à la base de données.

D'autres types d'EJB existent (EJB message, JMS) mais ne seront pas abordés dans ce TP.

Table des matières

1 Péparation

1.1 JBoss

Pour exécuter des EJB il est nécessaire d'avoir un conteneur d'EJB. C'est le rôle du serveur d'applications. Dans ce TP, nous utiliserons JBoss (projet opensource). À l'IUT JBoss est installé (comme tomcat) dans /opt. Pour automatiser ou créer des scripts il peut être utilise de définir une variable environnement $JBOSS_HOME pointant sur le dossier de JBoss. Ou encore $JBOSS_DEPLOY.

```
# Dans le fichier .bashrc
# IUT
export JBOSS_HOME=/opt/jboss....
# @Home
export JBOSS_HOME=/home/toto/workspace/java/JBoss

export JBOSS_DEPLOY=$JBOSS_HOME/server/default/deploy
```

Questions
- Quel est la différence entre Tomcat et JBoss ?
- Démarrez JBoss avec le script bin/run.sh.

1.2 Les fichiers *Enterprise ARchive*

Un EAR (pour Enterprise Application ARchive) est un format de fichiers utilisé pour empaqueter un ou plusieurs modules Java EE dans une seule archive, de façon à pouvoir déployer ces modules sur un serveur d'applications en une seule opération, et de façon cohérente. Ces archives contiennent aussi des fichiers XML de configuration

ou des fichiers appelés “descripteurs de déploiement”, qui indiquent comment les modules doivent être déployés sur le serveur.

Dans ce TP, nous allons créer un projet EAR contenant :
- Un fichier JAR qui contiendra tout le code de la logique métier de notre application (les EJBs)
- Un fichier WAR qui est le module WEB qui interrogera la logique métier.

Il est bien entendu possible de mettre plusieurs fichiers WAR et plusieurs fichiers JAR au sein d'un même EAR.

Comme pour les fichiers WAR, nos fichier EAR vont être généré par Maven. Néanmoins, il existe souvent des dépendances entre les projets Web (WAR) et les projets EJB (JAR). Ainsi, pour pouvoir compiler un projet WAR disposant d'une dépendance, il sera au préalable nécessaire d'installer (dans le dépôt Maven) le projet EJB. À l'IUT, ce dépôt Maven est partagé (pour les quotas utilisateurs) et ce situe dans /net/opt/Maven/repository. Chez vous ou sur une autre machine le dépôt Maven est situé en générale dans $HOME/.m2

Questions
- À quoi sert la séparation de la logique métier et la partie Web du coté serveur ?
- Comment installer les différents sous-projet d'un projet Maven ?
- Comment générer un EAR ?

Nos EJBs vont être exécutés dans un serveur d'application appelé JBoss. Pour cela il faut copier-coller le fichier EAR vers $JBOSS_HOME/server/default/deploy. L'ors du déploiement, JBoss affiche de nombreux messages de logs. Il n'est pas nécessaire à votre niveau d'en comprendre l'intégralité, mais il est absolument nécessaire de les lire et de pouvoir détecter une erreur de déploiement.

1.3 Récupération du projet squelette

1. Récupérez le projet Maven à l'adresse suivante (Attention : pas de *cp*) :

Source 1 – Depuis chez vous

```
hg clone ssh://your_login@info-ssh1.iut.u-bordeaux1.fr//net/
   Bibliotheque/JEE/2011-2012/tp2-ejb/skull-tp-ejb/
```

Source 2 – A l'IUT

```
hg clone /net/Bibliotheque/JEE/2011-2012/tp2-ejb/skull-tp-
   ejb/
```

2. Installez les différents projets dans votre dépôt Maven (*mvn clean install*).
3. Générez des projets eclipse et ouvrez-les.
4. Parcourez les 4 fichiers pom.xml. Faites un diagramme de dépendance entre les projets (qui à besoin de qui).
5. Générez un fichier EAR
6. Déployez-le sur JBoss et tester que votre EAR est bien déployé.

2 Développement de l'EJB Entity : "Contact"

Le projet EJB ne contient pas de dossier de sources pour les classes Java. Rajoutez un nouveau dossier de sources appelé *java* dans le dossier */src/main/*. Afin de simplifier le partage des tâches dans le cadre du développement d'un projet, chaque couche est identifiée par un package. Par exemple, le package *iut.jee.contact.entities* contiendra les beans du projet et *iut.jee.contact.services* contiendra les services.

Questions

1. A quoi servent les packages Java ? Pourquoi est il considéré comme "*sale*" de ne pas les utiliser ?
2. Ajoutez une nouvelle classe (Java Bean) nommée "Contact" dans le package correspondant.
3. Complétez cette classe avec les attributs et méthodes nécessaires qui doivent correspondre à la classe Contact du TP précédent. Pour plus de simplicité, vous pouvez laisser de côté les attributs de type *Date*. De plus, vous pouvez utiliser les raccourcis d'éclipse (*generate getters and setters* par example)
4. Remplacez la classe Contact du *WAR* avec cette nouvelle classe (ie : supprimez l'ancienne classe Contact).
5. Ajoutez les annotations Java permettant de "transformer" cette classe en tant qu'entité (cf. Fig. 2) : on annotera la classe en tant qu'Entity (*@Entity*) et on indiquera son attribut clé (*@Id*). L'annotation *@Table* indique le nom de la table qui stockera cette entité.
6. Le projet ne compile pas. Pourquoi ?
7. Ajoutez les dépendances Hibernate afin de compiler la classe. Il est nécessaire de *signaler* à eclipse que les dépendances du projet ont changés. Pour cela utilisez la commande *mvn eclipse :eclipse -Dwtpversion=2.0* et rafraîchissez ensuite le projet depuis eclipse.
8. Le projet ne compile toujours pas ? C'est normal, il faut importer les différentes annotations Java dans le fichier Contact.java (directive *import ...*) (Ctrl+Shift+O)
9. Déployer une nouvelle version de l'EAR (n'oublier pas le *mvn install*).
10. Dans votre navigateur, ouvrez la page : http ://localhost :8080/ => JMX Console => JBoss : database=localDB,service=Hypersonic. Puis dans startDatabaseManager clickez sur Invoke. Que constatez-vous ?

```
[...]
@Entity
@Table( name = "Contact" )
public class Contact implements Serializable{

        @Id
        @Basic
        private String name;

        @Basic
        private String firstname;
}
```

FIGURE 2 – Exemple d'un EJB Entité.

Questions

1. Expliquer chacune des annotations Java présente dans la classe Contact.
2. Quel annotation faudrait-il ajouter pour ajouter un attribut de type *List<Adresse>* à la classe contact ?

3 Développement de l'EJB Session : ContactService

Dans un souci de maintenir la cohérence des objets manipulés, la conception de projets JEE exige de ne pas exposer directement les EJB Entité aux clients. Pour gérer et assurer la cohérence des données, nous utilisons le patron de conception (design pattern) *Façade*, qui a pour but d'exposer des services aux clients par l'intermédiaire d'une interface distante pour la manipulation de beans.

Ici nous allons créer cette interface (i.e. : ContactService) puis la classe qui l'implémente. L'interface définit les méthodes à exposer aux clients.

Nous allons pour l'instant nous concentrer sur l'ajout d'un contact. L'interface ContactService n'aura donc, pour l'instant qu'une méthode *Contact add(Contact c)*.

3.1 Création de l'interface du bean

1. Ajoutez une nouvelle interface au projet nommée *ContactService* (package *contact.services*).
2. Complétez l'interface avec les méthodes nécessaires.
3. A quoi sert cette interface ?
4. À quoi sert l'annotation *@Remote* ?

```
[...]
@Remote
public interface ContactService {
        public Contact add(Contact e);
        [...]
}
```

FIGURE 3 – Exemple d'un EJB Session.

3.2 Implémentation de l'interface du bean

1. Ajoutez une nouvelle classe implémentant l'interface précédente.
2. Ajoutez les annotations nécessaires ; c'est notamment dans cette classe que le choix est fait entre un service dit "sans état" (*@Stateless*) et "avec état" (*@Stateful*).

```
[...]
@Stateless
public class ContactServiceImpl implements ContactService {

        [...]
}
```

FIGURE 4 – Exemple d'un EJB Session.

Questions

1. Lors du re-déploiement de l'EAR, que constatez-vous dans les messages de logs de JBoss ?
2. Quelle est la différence entre @Stateless et @Stateful ? Donnez des cas d'usage.

Mise en base de donnée

Lors du déploiement, le serveur d'application groupe les EJB Entité dans des unités de persistance. Chaque unité de persistance doit être associée à une source de données. L'*EntityManager* est le service qui permet de gérer ces unités. Le fichier *persistence.xml* est le descripteur de déploiement qui contient la configuration de l'EntityManager.

1. Modifier la classe *ContactServiceImpl* pour qu'elle utilise une instance d'un objet *EntityManager* qui se chargera d'interagir avec la base de données (cf. Fig. 5).
2. Ouvrez le fichier persistence.xml dans src/main/resources/META-INF.

3. Quelle est la relation entre les annotations que nous venons d'ajouter et le fichier persistence.xml ?
4. Ouvrez le fichier $JBOSS_HOME/server/default/deploy/hsqldb-ds.xml. Quel est le lien avec le fichier persistance.xml ?
5. À quoi correspond un *Data-source* comme DefaultDS.
6. Si l'on voulait utiliser une base de donnée MySQL, que devrait on créer et modifier ?
7. Quel est donc l'intérêt finale du fichier *persistance.xml* ?
8. Quel est l'avantage de la base de donnée décrit par le data-source *DefaultDS* ?

```
[...]
@Stateless
public class ContactServiceImpl implements ContactService {

        @PersistenceContext(name="MyEntity")
        private EntityManager em;

       public Contact ajouter(Contact e) {
              em.persist(e);
              return e;
       }
[...]
```

FIGURE 5 – Exemple de modification de la classe d'implémentation du service.

4 Communication entre le WAR et les EJBs

Utiliser un service EJB depuis un WAR

1. Créez une méthode (*getContactService()*) pour récupérer une instance de ContactService en vous inspirant du code 6. Cette méthode utilise un protocole appelé RMI que nous verrons dans le prochain TP.

Questions

1. Commentez la fonction *getContactService()*.
2. A quoi correspond la chaîne *MonDynamicProjectEAR/MonServiceImpl/remote*
3. Où la trouve-t-on ?
4. Pourquoi utilise t'on cette méthode plutôt q'une instanciation basique (*new MonServiceImpl()*) ? Donner des cas d'usage ou le *new* ne pourrais pas fonctionner.
5. Modifiez le projet ContactWeb afin d'ajouter un contact en base de données. Compilez, déployez, puis, vérifiez que tout fonctionne et que le nouveau contact est bien ajouté en base de données.

```
Context context;
MonService s = null;
        try {
                context = new InitialContext();

                s = (MonService) context
                                .lookup("MonDynamicProjectEAR/
                                   MonServiceImpl/remote");
        } catch (NamingException e) {
                e.printStackTrace();
        }
```

FIGURE 6 – Récupérez une instance d'un service EJB depuis le context initial.

5 Une Application CRUD

Implémentez toutes les fonction nécessaires dans le projet WAR et dans l'EJB afin de faire une application CRUD (Create, Retreive, Update, Delete) :

- Lister des Contact, et pouvoir voir le détail d'un Etudiant.
- Ajouter des Contacts en base de donnée.
- Modifier des Contacts
- Supprimer des Contacts

6 La culture Java EE (ou : avoir une bonne note à l'exam)

À l'aide de votre navigateur web favori, répondez aux questions suivantes :

1. Qu'est-ce qu'un serveur d'applications ? Citez un exemple.
2. Qu'est-ce qu'un fichier EAR ?
3. Qu'est-ce que JPA ?
4. Qu'est-ce que JPAQL ? Quelle est la différence entre JPA Criteria API et JPAQL ?
5. Qu'est-ce que Hibernate ? Quelle est la relation entre hibernate et JPA ?
6. Qu'est-ce qu'un EJB Entity, un EJB Session ?
7. Quelle est la différence entre @Stateless et @Statefull ?
8. Avec la classe EntityManager, comment effectuer une recherche par critère ?
9. D'après vous quels sont les avantages à utiliser des EJB Entity ?
10. Que faut-il faire pour utiliser une base de données MySQL au lieu de la base de données de JBoss ?
11. Qu'est-ce que NoSQL ?
12. Qu'est-ce que Spring Framework ?