

TP JEE

Contact Book

Nous allons travailler sur le projet Contact Book.

Prérequis MySql

1. Connectez vous à l'url <https://services.emi.u-bordeaux1.fr/dbserver>
2. Activer votre base de données mySql.
3. Dans votre navigateur, connectez-vous à votre base avec phpMyAdmin sous grincheux. Lien : <https://services.emi.u-bordeaux1.fr/dbserver/phpmyadmin/>.
Observez la base de données et les tables présentes ou non.

1 Prise en main

1. Récupérez le dossier TPJEE, déplacer le dossier jboss-4.0.4.GA dans C:/
2. Décompressez l'archive, lancez Eclipse (si besoin changez de workspace)
3. Changez de perspective pour passer en perspective J2EE.
4. Familiarisez-vous avec l'arborescence et les fichiers à votre disposition (Project Explorer).
5. Constatez un certain nombre d'erreurs (onglet Problems). Ces erreurs interviennent lors du premier déploiement de l'application (serveur et client) : il faut faire le lien avec les différentes librairies de notre environnement, notre source de données...
6. Commençons par les librairies : Faîtes les différents liens avec les librairies de jboss (jboss-ejb3.jar, jboss-ejb3x.jar, ejb3-persistence.jar, hibernate3.jar, antlr-2.7.6.jar), de axis (axis.jar, axis-ant.jar, commons-discovery-0.2.jar, commonslogging-1.0.4.jar, jaxrpc.jar, log4j-1.2.8.jar, saaj.jar, wsdl4j-1.5.1.jar). Pour cela, attardez-vous sur les propriétés des projets (serveur et client) : ContactBookServices & ContactBookClient.
(Project ! Properties puis Java Build Path et modifiez les variable JBOSS_HOME & AXIS_HOME) .
7. Nous allons faire le lien avec le seveur : double-click sur JBoss. Dans Server Overview, éditez le serveur, puis configurez-le (il est important d'éditer le chemin avant de configurer le serveur). Sauvez les changements.
8. Analyser puis modifier le fichier contactbook-ds.xml présente sous le repertoire server/default/deploy de jboss.
9. Lançons le serveur : start. Au démarrage, l'application ContactBook est déployée : les EJB et services web associés sont rendus accessibles, et les tables de la base de données pour la persistance sont créées si elles ne sont pas déjà présentes.
10. Dans votre navigateur, connectez-vous à votre base avec phpMyAdmin. Lien : <https://services.emi.u-bordeaux1.fr/dbserver/phpmyadmin/>. Observez la base de données et les tables présentes.
11. Éditez la classe de test ContactBookServiceEJBTestCase.java dans ContactBookClient/junit/contact.book.services.ejb. Cette classe de test permet en l'état d'ajouter un contact en utilisant l'EJB déployé sur le serveur d'applications. Vous pouvez

personnalisez l'entrée.

12. Puis lancez la classe de test : Run as JUnit Test.

13. Observez le résultat dans la base de données.

14. Nous allons maintenant utiliser le service web associé à la méthode addContact.

Analyser et modifier le fichier build.xml.

15. Éditez la classe de test ContactBookServiceTestCase.java dans ContactBookClient/junit/contact.book.services.jaws. Cette classe de test permet en l'état d'ajouter le contact ("Web", "Services", "Web@Service.com", "france") en se basant sur une Web method. Vous pouvez personnaliser l'entrée.

16. Puis lancez la classe de test : Run as JUnit Test.

17. Observez le résultat dans la base de données.

18. Tracez l'exécution d'une insertion via web service et d'une insertion via EJB (Différence de date avant et après l'exécution). Analyser et expliquer le résultat.

2 EJB Serveur

Nous allons maintenant créer de nouveaux services dans l'EJB accessibles directement par le Client Java et par Web Method.

1. Dessinez le schéma des classes : couche d'accès aux données, définition d'un contact, proposition des services, implémentation des services.

2. En vous inspirant du service addContact, proposez un service addCountry qui permet d'ajouter dans la table country un pays et son zip-code. Ce service sera disponible pour l'EJB et par Web Method.

3. Modifiez les différentes classes de test et testez ce nouveau service (par EJB et Web Method).

TP JEE (Seconde Partie)

Contact Book

1 EJB

1. Examinez le code serveur et l'implémentation de addContact. Déterminez le rôle de chaque classe.

2. Implémentez et testez la fonction :

```
public Contact getContact(Integer idContact)
```

Mots clefs : méthode find pour l'entité manager em.

3. Implémentez et testez la fonction :

```
public void printInfoContact(Contact contact)
```

qui affiche à l'écran dans la console toutes les informations d'un contact.

4. Implémentez et testez la fonction :

```
public void removeContact(Integer idContact)
```

Mots clefs : méthode remove pour l'entité manager em.

5. Implémentez et testez la fonction :

```
public void updateContact(Contact contact)
```

qui met à jour les informations d'un contact (nom, prenom, mail, pays).

Mots clefs : méthode merge pour l'entité manager em.

2 WebMethod

Nous allons enrichir maintenant le service web qui ne contient actuellement que la méthode `addContact` (annotation : `@WebMethod`).

Le client (java, C#, php) n'a pas connaissance de la classe `Contact` : Observez donc les différentes signatures de la méthode `addContact`.

1. Pour chaque fonction implémentée ci-dessus, est-il intéressant de la proposer en Web Service ; et si oui, avec quelle signature ?
2. Implémentez et testez vos différentes méthodes web.

3 Pour aller plus loin

À propos de `Contact` :

1. Implémentez et testez la fonction (non web) :

```
public List<Contact> getAll()
```

qui récupère la liste des contacts.

Mots clefs : méthode `createQuery` pour l'entité manager em.

2. Proposez vos propres méthodes (web ou non).

À propos de `Country` :

1. Écrivez la méthode `addCountry`. Testez-la.

2. Nous voulons maintenant proposer le web service associé. Pour cela, réalisez les étapes suivantes :

(a) Ajoutez l'annotation

```
@WebService(servicename = "CountryService")
```

(b) Écrivez la méthode web.

(c) Sur le client, éditez le fichier `Build.xml` pour prendre en compte la WSDL associée au service de `Country`.

(d) Exécutez-le afin de créer les classes java associées.

(e) Éditez la classe de test.