Rapport Projet Services Web Master 2 Informatique

Mateusz SZCZECH, Mael HANQUEZ, Lilian BERRUET

1. Les choix de conception

Le projet est divisé en deux parties:

- RMI avec le service EiffelBikeCorpService et le client EiffelBikeCorpClient se trouvant dans le répertoire EiffelBikeCorpProject
    - EiffelBikeCorpService qui gere la location des vélos
    - EiffelBikeCorpClient qui communique avec EiffelBikeCorpService pour pouvoir louer des vélos aux utilisateurs
- Web service qui regroupe deux services et un client:
    - GustaveBikeService qui gere l'achat des vélos
    - BankService qui gere les fonds des utilisateurs
    - GustaveBikeClient qui permet de communiquer avec GustaveBikeService pour acheter des vélos

a) La partie RMI

La communication entre EiffelBikeCorpClient et EiffelBikeCorpService se fait grâce aux interfaces dans le package "common".
EiffelBikeCorpService implémente deux interfaces:

- EiffelBikeCorpInterface qui permet de louer des vélos, de les rendre et de lister les vélos à loué, c'est cette interface qu'utilise EiffelBikeCorpClient
- EiffelBikeCorpAccessInterface qui permet d'avoir accès a un velo spécifique et de retirer des vélos. En effet on a besoin de ces fonctionnalités pour que GustaveBikeService puisse retirer les vélos que l'utilisateur veut acheter sans pour autant de donner accès a ces méthodes à EiffelBikeCorpClient

b) La partie Web Service

GustaveBikeService communique directement avec EiffelBikeCorpService grace aux interfaces du package "common" (qui a été copié de la partie RMI) mais aussi avec BankService pour savoir si le client peut ou non acheter un vélo et avec FxtopServices qui est un service en ligne permettant d'echanger la monnaie en temps réel.

GustaveBikeClient communique avec GustaveBikeService pour pouvoir acheter des vélos dans la monnaie souhaitée, rajouter des fonds, constituer son panier, afficher les vélos disponibles pour l'achat.

2. les difficultés rencontrées

a) les difficultés de la partie RMI

On n'a pas rencontré de difficultés particulières pour la partie RMI.

a) les difficultés de la partie Web Service

On a rencontré plusieurs difficultés pour la partie Web Service:

- La compréhension de comment fonctionne eclipse pour crée un web service et un web service client
- Quand on essaye de crée un web service client, eclipse generais des classes dans un package. Le code généré possédait des conflits d'imports. Le module se trouvait à la fois dans les jars et dans java.utils.xml. On a dû changer la version de java en JEE 1.5 ce qui a réglé le souci
- Pour la création des méthodes dans GustaveBikeService, eclipse ne nous laissait pas de générer le fichier .wsdl sans donner l'erreur précise. Apres 4h d'essais on a compris qu'on n'a pas le droit d'utiliser de lambdas, de return seulement des tableaux et pas de list et qu'il fallais remplir le tableau a la main et pas avec la méthode list.toArray()
- La sérialisation de la classe GustaveBike ne fonctionnait pas, grâce à votre aide on a compris que les noms de nos packages commencés par une majuscule ce qui causait le problème
- Le maintien de la session avec GustaveBikeService et BankService ne fonctionnait pas, à chaque fois une nouvelle instance du service a été crée, il fallais rajouter à la main le "ns1:parameter" avec le "name: scope" et la "value: application" dans le server-config.wsdd
- Il fallais faire attention de ne pas avoir déjà un serveur lancé sur le port 8080, une personne d'entre nous avais spring lancé sur ce port qui empêchait le lancement du serveur avec les services

2. Manuel d'utilisation

a) Pour la partie RMI

Tout d'abord mettez vous dans le repertoire racine du projet: EiffelBikeCorp

Lancez le service EiffelBikeCorpService qui permet de louer des vélos avec la commande:

```
java -classpath EiffelBikeCorpProject/out/production/EiffelBikeCorpProject
EiffelBikeCorpService.EiffelBikeCorpService
```

Quelques vélos vont etre rajoutées par défaut dans le service.

Ensuite vous pouvez lancez autant de clients de EiffelBikeCorpService que vous voulez (1 terminal = 1 client) avec la commande:

```
java -classpath EiffelBikeCorpProject/out/production/EiffelBikeCorpProject
EiffelBikeCorpClient.EiffelUsersClient
```

Pour lancer le client en interface graphique crée avec Spring Boot:

Mettez vous dans le repertoire EiffelBikeCorpSpring et lancez les deux commandes:

```
mvn clean package
mvn spring-boot:run
```

Maintenant ouvrez votre navigateur et tapez dans l'URL:

```
localhost:8081
```

La partie rajout dans la queue n'est pas implementé dans l'interface spring graphique mais seulement en terminal.

b) Pour la partie Web Service

Tout d'abord mettez vous dans le repertoire racine du projet: EiffelBikeCorp

Si ce n'est pas déja fait lancez le service EiffelBikeCorpService qui permet de louer des vélos avec la commande:

```
java -classpath EiffelBikeCorpProject/out/production/EiffelBikeCorpProject
EiffelBikeCorpService.EiffelBikeCorpService
```

Rendez le script catalina.sh du serveur tomcat 8.5 executable

```
chmod +x apache-tomcat-8.5.84/bin/catalina.sh
```

Lancez le serveur avec les Web Services: GustaveBikeService et BankService

```
./apache-tomcat-8.5.84/bin/catalina.sh start
```

Pour arreter le serveur:

```
./apache-tomcat-8.5.84/bin/catalina.sh stop
```

Pour lancer le client GustaveBikeClient qui permet de communiquer avec GustaveBikeService

```
java -jar GustaveBikeClient.jar
```