Antoine Carat Margaux Cherrueau Thibaud Courtoison Maël Quémard
23 décembre 2017

Rapport Middleware

1. Pourquoi le projet 3 ?

Nous avons commencé par parcourir les 3 projets en lisant leur rapport et en regardant rapidement leur code. Nous avons commencé par éliminer le premier projet (pay2bid) qui pour nous était un peu trop bien réalisé et ne nous laissait pas de marge de manoeuvre. Nous avons alors choisi le projet 2 (coin-coin), nous trouvions le projet assez clair et mieux documenté que le projet 3. Cependant à cause des dépendances Maven, Tomcat, … Nous n'avons jamais réussi à lancer le projet. Nous avons donc choisi le projet 3.

2. Les mauvais points

Nous avons commencé par exécuter le programme pour ensuite parcourir le code plus en détail et nous avons trouver quelques points que nous trouvions non adaptés à la solution. Voici la liste de nos remarques.

- Général :
  - La structure du code n'était pas très bien structurée.
  - Une salle d'enchère (1 article à la fois, round par round à l'aveugle) plutôt d'un eBay (plusieurs ventes à la fois).
  - La gestion des exceptions un peu limitée.
  - Partage d'objet concrets, un objet n'est pas une interface.
  - Il n'y a pas de persistance des données.
- Client :
  - Il faut obligatoirement attendre qu'il y est 2 clients pour pouvoir lancé une enchère.
- Serveur
  - Revoir l'interface *Vente*.
- Chrono
  - Une classe *Chrono* sans grand intérêt.
  - C'est le client qui est composé d'un chrono plutôt que l'article en lui même.
- Objet
  - L'Objet *Objet* (l'article en vente) n'est pas assez développer.
  - Les articles déjà présent dans la base de données était créer par le client plutôt que d'être créé par le serveur puis envoyé au client
- IHM
  - Le processus Client ne se ferme pas lorsque nous fermons la fenêtre
  - L'IHM pourrait être un peu plus évolué.

3. Nos changements

Pour palier aux mauvais points que nous avions trouvé nous avons fait les modifications suivantes.

Nous avons commencé par restructurer le code, pour cela nous avons utilisé plusieurs packages Client.app contenant les classes métier du client (ClientApp, IClient et Item et SellableItem qui est une implémentation de Item qui est sérialisable. Nous ensuite le package Client.vue contenant l'IHM du client. Puis pour finir le package Serveur contenant le classe métier du serveur (IServer, ServerApp et DBManager) . Nous avons aussi tout mit en anglais pour avoir un projet plus exploitable.

Dans le projet de base nous avions du partage d'objet concret or cela rompe un des deux concepts fondamentaux de l'ingénierie logicielle,le couplage faible. Nous avons donc ajouté des Interfaces pour palier à ce problème. Ainsi nous n'avons plus de partage d'objet concret mais seulement des interfaces partagés.

Nous n'aimions pas vraiment cette idée de salle au enchère et nous voulions nous rapprocher d'un site de vente aux enchères tel que eBay avec des ventes simultanées, ou l'on peut enchérir ou non selon nos envies. De plus, dans le projet de base étant données que c'était une salle d'enchère il fallait obligatoirement attendre qu'il y ai au moins deux clients. Avec un site d'enchères nous n'avons pas besoin de cette contrainte.
Nous avons donc fait un serveur qui est toujours connecté et lorsqu'un client qui possède un pseudo unique se connecte à la plateforme, le serveur lui envoie tous les articles présents dans la base de données.

Le Client arrive sur interface composées de plusieurs onglets "Soumettre un article", "Mes achats" et "Enchères".

Lorsqu'il est dans l'onglet enchères, il peut voir les articles en cours ainsi que les articles déjà vendus. Sur un article en cours le client peut cliquer sur le bouton enchèrir et cela fera une augmentation de 20 % du prix ou bien il peut choisir la somme qu'il veut rajouter en changeant le champs de texte.

Dans l'onglet mes achats, le client peut retrouver tous les articles qu'il a acheté sur cette plateforme.

Et enfin dans l'onglet Soumettre un article, le client à la possibilité de vendre un article. Pour cela il a juste a donner le nom, la description, le prix ainsi que de la durée de la vente en minutes. Nous avons fait en sorte que le client ne voit pas les objets qu'il a lui même soumis car le client ne veut surement pas racheter son propre article.

Pour finir avec l'ihm du client nous avons en sorte que lorsque nous fermons la fenêtre le processus client s'arrête ce qui n'était pas fait dans le projet de base.

Chaque article est composé d'un nom, d'une description, d'un vendeur, d'un acheteur, d'un prix, d'un boolean pour savoir si oui ou non l'article est vendu ainsi que d'une date de fin de vente. Lorsque le client soumet son article et qu'il ajoute une durée de vente en minutes. Le serveur récupère cette information et calcul a quel date précise l'enchère se finit. Il ajoute ensuite l'article a l'onglet enchère. Ainsi lorsque le temps est écoulé le serveur ferme la vente pour cette article. Grace a cela nous avons enlever la classe Chrono qui était présente et à partir de maintenant c'est l'article qui est composé d'un chronomètre et non le client.

Pour finir nous avons ajouté une pseudo base de données dans un fichier JSON, nous permettant de garder les enchères et ainsi chaque clients peut se connecter et se déconnecter sans pour autant perdre ses informations. A partir de notre base de données, nous avons pu palier a un des problème que nous avions énumérer au début de projet. En effet nous avion remarquer qu'un article déjà présent dans la base de données était créer par le client plutôt que d'être crée par le serveur. Avec notre base de données, le serveur crée les articles puis les envoie aux différents clients.

4. Utilisation du programme.

1/ Ouvrir le projet dans une version d'éclipse >=Neon 3.
2/ Configurer la base de données :
Dans le constructeur de ServerApp, l'appel au constructeur de DBManager permet de configurer : (a) la recréation de la base de données (true/false)
(b) le pré-remplissage de la base de données. (true/false)
3/ Lancer ServerApp une fois.
4/ Lancer autant ClientApp autant de fois que de client souhaités.
Après avoir rempli le pseudo, vous accédez à plusieurs espaces :
(a) La soumission d'un nouvel objet
(b) La consultation de ses achats
(c) Les enchères, avec la possibilité d'enchérir de 20% du prix courant par défaut ou d'une somme supérieure.

5. Améliorations possibles.

Il aurait été intéressant de permettre à un utilisateur de se connecter à l'aide d'un mot de passe. Ainsi, deux utilisateurs utilisant un même pseudo ne pourrait accéder qu'aux informations qui leur sont propres.

L'application n'a pas été testée sur deux postes physiquement différents, même si en théorie cela devrait fonctionner, nous ne pouvons pas l'affirmer. Pour cela nous aurions pu faire 3 projets Maven correspondant aux trois packages actuels. Le projet *server* et le projet *client* dépenseraient ainsi du projet *shared*.

L'utilisation de Swing pour réaliser l'IHM est très contraignante, ainsi nous aurions pu utiliser une interface web réalisée en JavaScript et dialoguer avec le serveur via des web-sockets. Cela éviterai tous les problèmes liés à l'IHM comme les rafraîchissements « agressifs » (reconstruction de toute une partie de l'interface pour seulement un changement mineurs. Ces mises à jour causent par ailleurs des erreurs, qui ne sont pas fatales au programme mais qui existent tout de même.

6. Conclusion

Reprendre un projet existant est toujours un exercice difficile car il faut se plonger dans un code que l'on ne connaît pas, parfois mal écrit et/ou mal documenté. Cependant nous pensons avoir réussi l'exercice en améliorant le projet de base sans pour autant tout changer : Même si tout le code a été réécrit, nous nous sommes basé sur le code existant, notamment pour les interfaces IClient et IServer.