Middleware
Critique des projets

M2 ALMA — 2018-2019
Florent Alapetite - Roxane Bellot - Alexandre Boudine

Ce projet est hébergé sur GitHub

Pour ce projet de Middleware, nous devions analyser deux projets réalisés par des étudiants de promotions précédentes et choisir l'un des projets dans le but de lui apporter des modifications.

Problèmes observés

Gringotts

Nous avons en premier lieu découvert plusieurs problèmes liés à l'interface graphique :

- La mise en page est approximative et peu lisible
- Pas de messages hors du terminal lorsque les types d'entrée sont incorrects.
- Lorsqu'un refresh s'effectue, l'onglet consulté change
- À la fin des enchères, l'item reste dans la liste des enchères, et le gagnant n'est pas notifié
- On ne peut enchérir que supérieurement au montant par défaut (20%) (Ce n'est pas indiqué en dehors du message dans la console)
- Il n'y a pas d'onglets "mes ventes" qui répertorie les objets vendus, leur prix de vente et leur gagnant
- Les montants des enchères pourraient être arrondis à deux chiffres après la virgule pour éviter les nombres rallongés
- On ne voit pas son propre nom
- Si des items ont le même nom, seul le premier est pris en compte dans l'interface

Ces problèmes étant graphiques, et non pas des problèmes de synchronisation, ils ne seront pas notre priorité.

Nous avons ensuite relevé des problèmes liés au fonctionnement des enchères :

- Certaines exceptions ne sont pas traitées, d'autres n'ont pas de source triviale
- Les clients sont identifiés par un pseudo, donc plusieurs clients avec le même pseudo concernent le même compte
- Le serveur est forcément host sur la boucle locale (impossible de le rendre disponible depuis l'extérieur).
- On peut mettre des champs vides dans "nom" et "description" lorsque l'on rend un article disponible à l'enchère.
- Les enchères ne finissent pas toujours à la fin de leur temps imparti lorsqu'un client a proposé un item
- Le méthode `action_performed` du client est annotée "synchronized" : c'est une exclusion trop forte pour la plupart des actions. Seule l'action "enchérir" devrait être critique et surtout être gérée dans un moniteur côté serveur.
- La liste des objets en vente est présente en mémoire (dans un objet `JsonObject` et dans un fichier `db.json`). Vu que la liste n'est pas persistante d'un lancement à un autre, dans l'état, la db ne sert à rien.

Pay2bid

De la même manière, nous avons en découvert plusieurs problèmes liés à l'interface graphique :

- Actualisation des vues lorsqu'une enchère se termine
- Lorsqu'une enchère est terminée, il n'y a rien pour indiquer qu'elle est close et la personne l'ayant remportée n'est pas notifiée
- Actualisation qui ne s'effectue pas tout le temps quand il y a 3 clients (1 vendeur et 2 acheteurs) : il faut attendre une modification du 2ème acheteur ou la fin du temps pour avoir le rafraîchissement du prix actuel de l'enchère
- Il serait intéressant de rajouter une option pour mettre un temps à l'enchère

Le projet comporte également des problèmes concernant le fonctionnement des enchères :

- Des exceptions qui ne sont pas gérées (lettres dans les enchères)
- Il y a des problèmes concernant la gestion de la queue d'enchère :
  - Si une enchère est en cours, aucun nouveau client ne peut se connecter au serveur
  - Si une enchère est lancée avant qu'il n'y ait de client, tout est bloqué car l'enchère ne peut se terminer et aucun client ne peut se connecter pour enchérir

Modifications

Nous avons choisi de modifier le projet Gringotts car nous avons trouvé que le code était le plus perfectible et nous avons observé moults problèmes sur lesquels travailler.

Notre travail est accessible sur notre dépôt GitHUB à cette adresse: https://github.com/FlorentALAPETITE/Gringott

Succinctement, voilà un résumé de nos améliorations :

- Une erreur côté client (en particulier des mauvaises valeurs dans les champs de formulaire) fait afficher une **boîte de dialogue** à la place d'un log dans la console
- Nous avons ajouté un **.gitignore**
- Le serveur compte maintenant un système de **log** basique (dans un fichier à la racine) qui est réinitialisé à chaque lancement
- Les **clients** sont identifiés par un **ID** et stockés dans des `HashMap` (ce qui permet d'avoir plusieurs clients avec le même pseudo -> `pseudo@ID`) (par le serveur du moins).
- Les **objets** sont également identifiés par un **ID** (permet plusieurs objets de même nom également).
- Nous avons découvert que suite à la publication d'un article les ventes ne pouvaient plus se finir, à cause d'une erreur de programmation. En effet, le serveur cherchait à déterminer qu'il était temps de mettre fin à une vente en itérant sans fin sur la liste des articles (un ajout dans cette liste provoquait donc une erreur). En plus de résoudre le problème, désormais, pour chaque ajout, un **thread** est lancé et est en attente jusqu'à l'heure de fin, où se charge de lancer la fermeture de la vente. Le serveur est donc beaucoup moins demandant en **ressources**.
- Nous avons fait des améliorations notables au niveau de l'interface graphique :
    - Nous avons ajouté un onglet "**mes ventes**" recensant les objets soumis et un onglet déconnexion pour héberger le bouton qui était avant dans la liste des articles
    - En plus de cela, nous avons beaucoup amélioré la manière dont l'interface graphique est gérée, en arrêtant de tout détruire et tout

ré-instancier pour chaque action. L'interface est maintenant gérée **dynamiquement** en fonction des données envoyées par le serveur

  - Nous avons fixé un nombre maximal de **chiffres après la virgule** pour les prix

  - Les événements reçus par les clients déclenchent des **modifications visuelles dynamiques** (notifications, couleurs, mise à jour de l'interface)

  - La gestion des événements lorsqu'un bouton est cliqué se fait désormais avec un **listener par action** et non plus par un gros listener qui englobe toutes les actions

- Nous avons fait du tri dans les méthodes déclarées `Synchronized` :

  - Désormais, seule la méthode permettant de faire une offre est **synchronisée**. De plus, elle est gérée au moyen d'un **moniteur** dans le serveur.

  - Les autres méthodes (connexion, déconnexion, ajout objet) ne sont **plus bloquantes** afin d'augmenter la disponibilité du serveur sans impacter les fonctionnalités disponibles pour les clients. Pour cela, nous nous sommes assurés qu'aucun scénario ne puisse causer de problème quant à l'exactitude des données partagées (objets en ventes) entre le serveur et les différents clients.

- La base de donnée (basique) qui était fournie a été entièrement refondue pour permettre la **persistance** des objets et la sauvegarde de l'historique des offres (offres en cours et offres terminées) lorsque le serveur est éteint.

- Un message d'erreur s'affiche désormais lorsqu'un client essaie de se connecter **sans serveur**.

- Au lieu d'avoir uniquement un endpoint `addItem` et de laisser le serveur itérer sur la liste des items pour les envoyer aux clients, nous avons mis un place un `addItems` qui **envoie la liste** complète. En plus d'être plus rapide et plus lisible, c'est aussi une source d'erreurs qui disparaît.

Améliorations possibles

- Nous nous sommes concentrés sur le cas où le serveur reste bien vivant. S'il venait à tomber, les erreurs ne seront pas du tout traitées.
- Certaines fonctionnalités non indispensables ne sont pas implémentées, telles que re-proposer à la vente un item qui n'aurait pas trouvé d'acheteur, ou la mise à jour en direct des items que nous vendons.
- La persistance des données pourrait se faire autrement que via un fichier JSON, qui ne permet pas de requêtes, et que nous sommes obligés de réécrire en intégralité à chaque mise à jour.
- Dans l'idéal, si le projet venait à rencontrer des utilisateurs, une refonte totale de l'interface graphique serait indispensable.
- L'utilisation d'un pattern **Observer** pourrait totalement convenir à ce projet pour la notification des clients lors d'un ajout / modification d'un item.
- Il nous avait paru également intéressant, l'idée de centraliser les items sur le serveur, pour ne pas avoir de copies à modifier localement sur chacun des clients. L'accès aurait été effectué simplement par des requêtes d'état actuel des items. Cependant il nous a semblé compliqué de le mettre en place notamment pour l'a mise à jour d'un item, qui nécessite que le client retrouve l'item à mettre à jour dans son interface.