PROGRAMMATION WEB SERVEUR

MÉDIATHÈQUE

CHELSEY MILLO 207 - FANGYUAN LISA YE 204 -
WILLIAM ZHANG 207

IUT PARIS DESCARTES 2020-2021

Sommaire

Présentation du projet

Le projet que nous avons eu à réaliser en trinôme consistait à développer un programme Java gérant la gestion de documents pour une médiathèque. Cette application contiendra alors un côté client ainsi qu'un côté serveur qui resteront en communication 24h/24h 7j/7j.

Toutefois, cette application ne fournit des services dédiés qu'aux abonnés répertoriés dans la médiathèque. Celle-ci offre ainsi la possibilité aux clients d'effectuer 3 opérations :

- La réservation d'un document qui permettrait de mettre un document de côté durant un temps donné
- L'emprunt d'un document
- Le retour d'un document

Fonctionnement de l'application

Pour faire fonctionner notre application, nous avons créé deux projets java :

- Un pour le Serveur
- Un autre destiné aux Client.e.s

Côté serveur

Pour le côté serveur, nous avons tenté de mettre en place une architecture évolutive en utilisant l'abstraction. Par exemple, pour éviter les redondances de code, nous avons ajouté une classe abstract Service, pour que tous les services puissent utiliser les mêmes ressources partagées (la liste des Abonnes et la liste des Documents). Le Décorateur que nous avons utilisé permet aussi d'ajouter plus facilement de nouveau service qui n'est pas encore prévu.

Nous avons notamment ajouté une classe abstraite Documents qui implémente l'interface Document. La classe DVDs hérite de la classe abstraite Documents, ce qui lui permet d'utiliser et Refactoring toutes les fonctions de la super classe. Cela permet ainsi de réécrire sur une fonction déjà existante en y ajoutant des spécificités supplémentaires selon nos besoins et aussi permet d'éviter de devoir réécrire tout le code. Le but de cela est de pouvoir, dans le futur, faciliter une extension de notre application vers d'autres types de documents.

De plus, afin de faire fonctionner des options telles que la vérification de l'âge des clients, nous avons intégré util.Calendar, dans le but d'utiliser Calendar pour obtenir les âges des abonnées.

Nous utilisons aussi Timer() et TimerTask() sous forme de classes anonymes pour calculer les dates et la durée de l'emprunt, de la réservation et du retour.

Côté client

Quant au côté client, nous avons créé une classe ApplicationClient.java qui à pour rôle de communiquer avec le côté serveur afin d'effectuer certaines actions et pour s'assurer de ce dialogue, des sockets seront envoyées dès que le client ait choisi d'effectuer l'une des trois fonctions : réserver , emprunter et retour.
Chacune de ces fonctions utilisent des ports qui lui sont initialement attribués et qui échangeront les données avec le service concerné.

Si le client souhaite réserver un document, il devra communiquer son numéro d'abonné ainsi que le numéro du document qui l'intéresse. Le catalogue lui sera au préalable affiché, bien évidemment. Le client aura un délai de 2h pour venir récupérer le document sans quoi celui-ci redeviendra disponible

Dans le cas d'un emprunt, le client effectuera le même processus qu'une réservation mais récupérera le document immédiatement.

Enfin, pour un retour, le client n'aura qu'une seule unique tâche qui est d'entrer le numéro du document.

Ressources partagées

Les ressources partagées étaient l'une des notions fondamentales à aborder dans ce projet.

Au cours de ce projet, nous avons appris l'importance qu'il y avait à intégrer des verrous à des objets afin que ces derniers soient sécurisés, c'est la raison pour laquelle nous avons utilisé l'instruction synchronized().

En effet, plusieurs clients peuvent demander le même document en même temps, ce qui signifie que plusieurs threads demandent simultanément l'accès au même espace d'adressage, et cela poserait problème dans le cas où nous n'aurions pas mis le synchronized(). Ce problème-là est donc à l'origine de la concurrence. Le but de cette méthode est d'assurer que les accès aux ressources partagées n'entrent pas en concurrence. Or, nous savons qu'avec un verrou posé, aucune action peut être effectuée par un autre thread de manière concurrente.

Dans notre cas, pour éviter qu'un document soit réservé ou emprunté en même temps par différents clients, nous avons pris l'initiative de mettre synchronized() dans tous les fonctions qui impacteraient l'état du document (réservation, emprunt et retour) afin de s'assurer que le traitement des demandes des clients soit traité dans l'ordre d'arrivée.
Le premier thread s'exécute correctement, les autres sont quant à eux mis en attentes. Si nous prenons l'exemple de deux abonnés qui demandent la réservation d'un même document, le premier thread fonctionnera correctement et ainsi réussira sa réservation, mais le deuxième déclenchera ResevationException, car l'état du document sera indisponible.

Conclusion

Difficultés rencontrées

Durant la réalisation de notre projet, nous avons rencontré de nombreux obstacles qui nous ont empêché d'avancer à la vitesse que nous espérons.

En effet, nous avons tout d'abord eu du mal à déterminer les points nécessaires à la création de l'application. L'implantation et la mise en œuvre des nouvelles notions apprises durant cette période ont été l'une des plus grandes difficultés pour nous. Des problèmes surgissaient par exemple lors d'une tentative d'emprunt effectuée par un deuxième client ou bien encore dans la communication entre le côté serveur et le côté client. De plus, les notions de sécurisation, de performance et d'architecture du code ont été difficiles à mettre en place.

Les éventuelles améliorations

Nous croyons que nous pouvons certainement améliorer notre programme. En effet, nous aurions pu essayer d'optimiser le code autant que possible de sorte qu'il n'occupe le moins d'espace et que l'application s'exécute plus rapidement. De ce fait, apporter des améliorations futures auraient été plus facile et le coût de la maintenance aurait été moindre.

Nous pouvons également rajouter des fonctionnalités supplémentaires comme le fait de nous-même emprunter un livre dans une autre médiathèque pour notre client ou encore un système de malus pour les utilisateurs ne rendant pas le livre à temps.

Bilan final

Malgré les problèmes rencontrés, nous avons toutefois réussi à rendre un projet fonctionnel. Une fois de plus, ce fut une bonne expérience où nous avons appris à travailler de manière autonome et à communiquer avec les autres pour trouver des idées. Mais ce projet nous a surtout permis d'apprendre et de comprendre comment la communication s'effectue entre le côté client et le côté serveur.