CHEVREUIL Cédric

EL GUETIBI Sofian

GAHLOUZ Jughurta

Service WEB et technologies REST

Projet Services Web

Rapport

M2 LID | S3 | 2022-2023

Table des matières

Introduction

L'objectif de ce projet est de proposer une application de location et d'achat de vélo.

L'application doit dans un premier temps permettre la location de vélo aux étudiants et aux employés de l'université Gustave Eiffel. Les vélos peuvent appartenir à l'université, ou bien à tout utilisateur du service. Tous les membres peuvent ainsi parcourir le catalogue de vélo et s'inscrire sur la liste d'attente afin de louer le vélo. Quand vient leur tour, une alerte leur est envoyée. A la restitution du vélo, l'utilisateur peut alors laisser un commentaire sur le vélo.

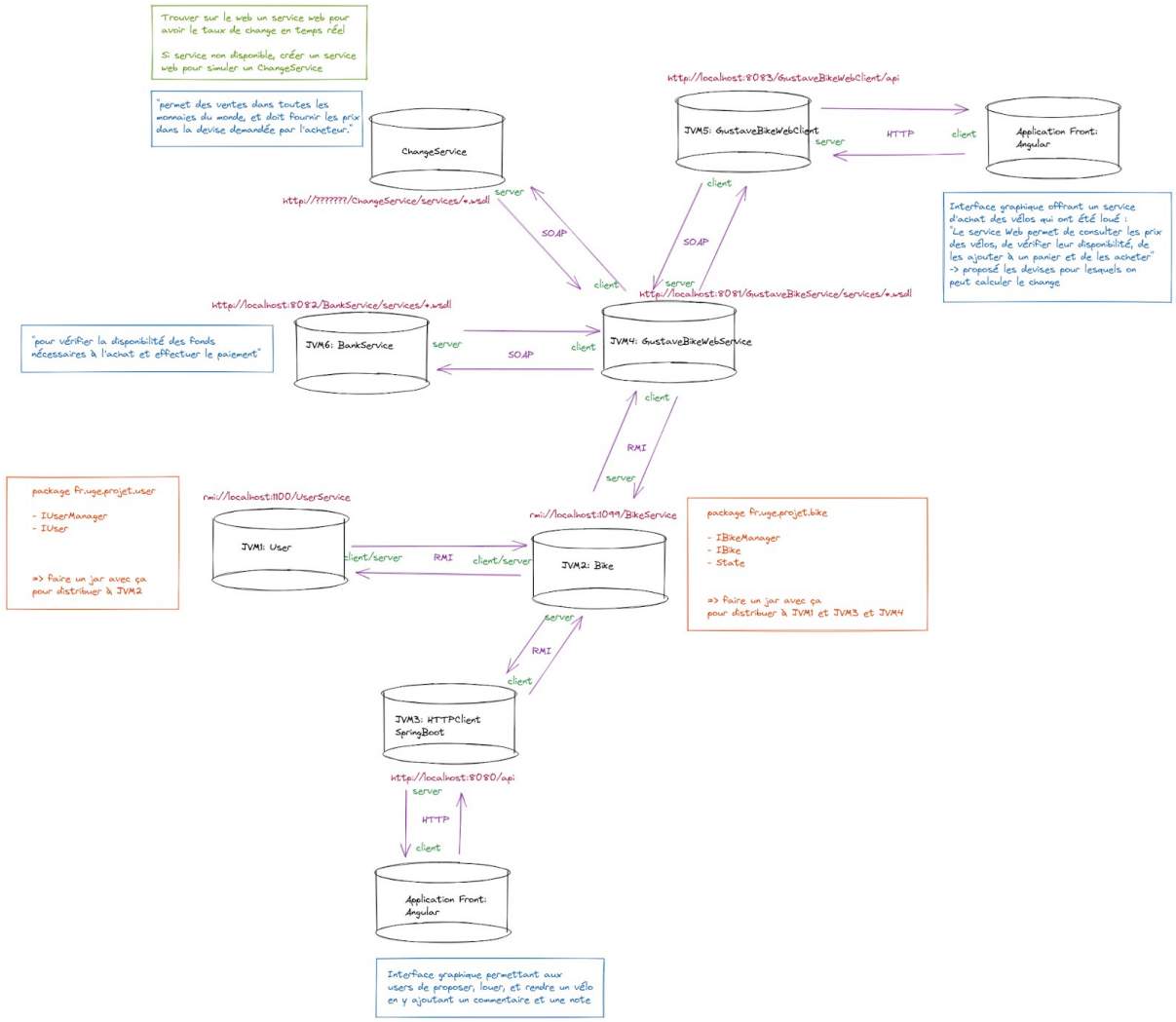
Dans un second temps, l'application doit également permettre à toute personne étrangère à l'université de pouvoir acheter les vélos présents dans le catalogue. Pour cela il peut consulter le prix, et les commentaires des vélos disponibles, et acheter celui qu'il veut. L'application se chargera alors de convertir la devise utilisée si nécessaire, et vérifier que la personne dispose bien des fonds nécessaires.

Le but de ce projet est d'architecturer cette application à l'aide de services communiquant les uns avec les autres à travers différents protocoles de communication: RMI, SOAP.

Choix de conception

Il a été très intéressant au début de notre projet de réfléchir à l'architecture que devra adopter notre projet. Le sujet nous à alors servi de cahier des charges afin de déterminer les fonctionnalités que notre application devra fournir. Nous avons ainsi pu identifier quels devaient être les différents services à mettre en place et quelles méthodes ils devront exposer.

A la fin de cette période de réflexion, nous avons pu convenir de l'architecture ci-dessous, que nous avons fait évoluer par la suite.



On retrouve dans la partie basse du schéma les services utiles à la location des vélos pour les étudiants et les employés de l'université, et dans la partie haute, les services bancaires et d'achat de vélos.

En partant du bas du schéma nous retrouvons la JVM2: Bike. C'est ellequi va gérer la partie métier propre aux vélos. Elle va ainsi stocker en mémoire les vélos louables qui ont été enregistrés, mettre à jour leur disponibilité, recueillir les réservations, et contacter les utilisateurs lorsque vient leur tour.A sa gauche, on trouve la JVM1: User, son rôle quant à elle est la gestion des

utilisateurs. C'est elle qui va stocker les utilisateurs, enregistrer les nouveaux, et vérifier leur existence.

Enfin, en dessous, la JVM3 représente l'utilisateur de l'application. C'est via cette JVM que l'utilisateur va pouvoir utiliser les services offerts par l'application. Il pourra ainsi: se connecter à l'application, consulter la liste des vélos présents au catalogue, s'inscrire dans la file d'attente d'un vélo.

Ces trois JVMs communiquent entre elles via une API RMI. Si un client veut se connecter à l'application, il doit communiquer avec la JVM User au travers de ses API. Une fois connecté, cette dernière lui fournira en réponse un objet User. Cet objet lui servira alors de jeton de connexion, et permettra au reste des services de reconnaître l'utilisateur à l'aide des données qu'il contient. Pour initier n'importe quelle opération pouvant modifier le catalogue, l'utilisateur devra lui fournir cet objet, le service vérifiera alors auprès de la JVM User que l'objet est valide au travers des API RMI fournies.

Dans la partie haute du schéma, on peut retrouver tous les services jouant un rôle dans l'achat de vélos par des utilisateurs étrangers à l'université. On retrouve ainsi quatres JVMs, chacune chargée d'une logique métier différentes et proposant une API SAOP afin d'être contactées par d'autres services.

Un service bancaire, cette dernière propose alors une interface permettant de vérifier qu'un usager dispose bien d'une certaine quantité d'argent sur son compte.

Un service de conversion de devise. Son interface expose ainsi une fonction permettant de convertir le montant d'une devise vers une autre.

Un service GustaveBikeWebService propose une interface SOAP permettant de consulter les vélos disponibles à l'achat, de recueillir des informations concernant un vélo en particulier et de l'acheter. Afin de

fonctionner, elle communique avec le service Bike présenté précédemment via une API RMI.

Pour finir, une JVM représente le point d'entrée des utilisateurs qui souhaitent profiter du service d'achats de vélos.

Au cours du développement, nous nous sommes demandés comment distinguer les utilisateurs du service de location et les utilisateurs du service d'achat. En effet, nous ne voulions pas proposer les méthodes propres à la location à ceux voulant acheter des vélos ou inversement. Pour ce faire, dans le service BikeManager (JVM5) nous avons séparé l'interface IBikeManager en deux sous interfaces, IBikeManager (lease) ne proposant que des méthodes pour louer, et IBikeManager (sell) ne proposant que des méthodes pour l'achat. Ainsi, suivant la JVM qui contacte ce service, BikeManager va renvoyer un objet implémentant l'une ou l'autre de ces deux interfaces.

Difficultés rencontrées

Tout d'abord nous avons eu du mal à faire communiquer notre service Bike avec notre service GustaveBikeWebService en RMI. Malgré les instructions présentes dans le cours, nous avons fait face à des erreurs. En effet, notre service GustaveBikeWebService n'est pas capable de trouver les classes implémentant l'interface qu'il souhaitait manipuler. Notre service n'arrive pas à chercher au bon endroit le codebase attendu.

Dans un second temps, nous avons également euent des difficultés à configurer une page web au-dessus de notre GustaveBikeWebClient. En effet, nous pensions utiliser Spring ainsi que ThymeLeaf afin de générer une page web contenant les informations reçues à partir des requêtes SOAP. Toutefois au moment de la configuration nous avons rencontré un conflit entre Spring et les Web Services.

Nous avons également eu du mal à envoyer un type complexe par le biais du protocole SOAP. En effet, dans la dernière partie du projet, il nous fallait envoyer des objets “Bike” à la JVM5: GustaveBikeWebClient, afin que l’acheteur puisse consulter la fiche d’un vélo. Il nous a ainsi fallu rechercher et comprendre le fonctionnement des fichiers wsdl, ainsi que l’assistant graphique, pour construire le type complexe qu’il nous fallait, et ainsi adapter la définition de notre opération.

Manuel d’utilisation

Pour la compilation de nos différents programmes nous nous sommes reposés sur différents outils de build. Pour la partie rmi nous avons utilisé gradle ainsi que maven pour l’application spring.

Pour lancer ses différentes applications un script bash, setup.sh, est présent à la racine du projet. Ce script lance les tâches gradle et maven responsable de la génération des 3 jar des 3 applications. Ensuite le script lance dans 3 terminals (gnome uniquement disponible sur linux) les trois programmes, et une dernière instruction ouvre firefox sur la page de connexion de l’application spring.

Pour la partie Service Web il faut ouvrir dans eclipse-jee le répertoire service-web dans lequel se trouvent les différents services web ainsi que les clients web.

Conclusion

Ce projet a été pour nous l'occasion de mieux comprendre comment fonctionne la communication entre les programmes, l'utilisation des protocoles

RMI et SOAP et l'architecture micro-service. Au cours de nos manipulations, nous avons pu découvrir les avantages ainsi que les limites des API que nous avons mises en place.

Nous avons ainsi pu voir qu'il était très facile de mettre en place le protocole RMI. Toutefois, cette dernière était assez contraignante à modifier. En effet, il est nécessaire de maintenir à jour à la main toutes les copies de l'interface. En ce qui concerne le protocole SAOP, il est plus rudimentaire et donc plus complexe à mettre en place. Il est en effet nécessaire de générer un bon nombre de fichiers dont un fichier wsdl résumant l'API. En revanche, son avantage et que n'importe quel langage de programmation sachant manipuler le wsdl, peut récupérer ce fichier et ainsi communiquer avec notre service.

Enfin, ce projet a été pour nous une bonne occasion de voir un autre aspect de l'informatique, l'architecture logiciel. Nous avons ainsi pu réfléchir en amont sur la manière de concevoir l'application, sur la manière dont allaient s'articuler les différents services, et comment ces derniers allaient communiquer entre eux.