Université Gustave Eiffel

Services Web et RMI

Rapport du projet

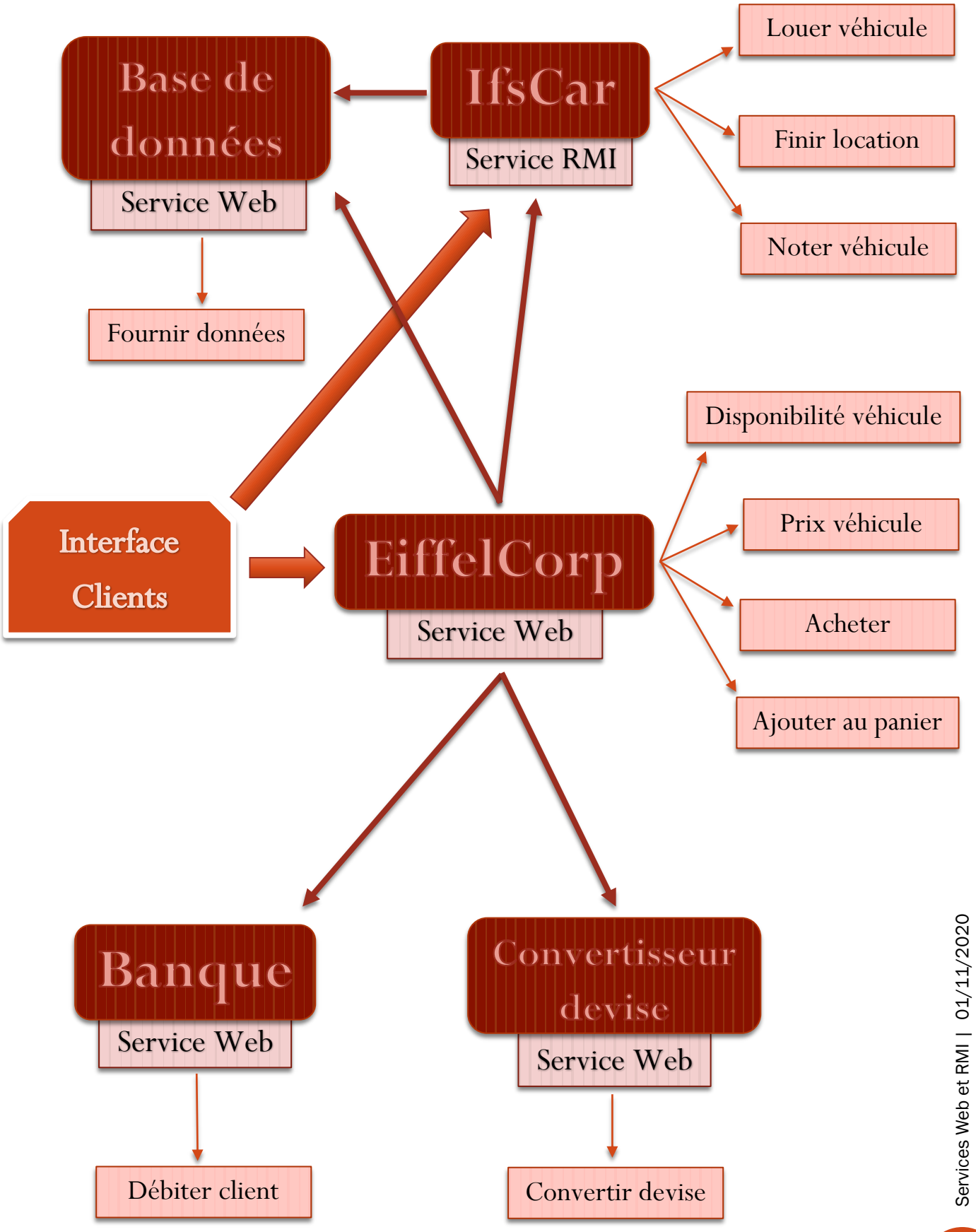
MASTER 2 LOGICIEL

01 novembre 2020

Samy WADAN – Akram MALEK – Amany MILED – Lamri KOUIDRAT

Table des matières

Architecture globale



Services

IfsCar – Service RMI

Le *RemoteObject* principal est l'objet `Garage`, il permet la location de véhicules en fournissant notamment la méthode `rent()` qui prend un objet de type `Employee` et l'identifiant d'un véhicule en paramètres.
Elle permet à un employé de louer un véhicule s'il est disponible, ou d'être placé dans une file d'attente sinon. Lorsque le véhicule devient disponible, l'employé est alors notifié en appelant sa méthode `notifyRented()`.
L'employé est donc également un *RemoteObject*.
Enfin, à la fin d'une location, l'employé peut noter le véhicule loué avec la méthode `Garage.grade()`.

Les véhicules disponibles par IfsCar ainsi que les notes des véhicules sont stockés dans une base de données.
Une classe `Vehicle` sérialisable contenant les caractéristiques d'un véhicule existe pour faciliter les échanges de données.

Convertisseur de devise – Service Web

Ce projet utilise un service web déjà existant permettant notamment de convertir un montant d'une devise à une autre, avec un taux de change actualisé en temps réel.
Le *wsdl* est disponible sur ce lien :
fx.currencysystem.com/webservices/CurrencyServer5.asmx?wsdl

Banque – Service Web

Le service web Banque permet de débiter un client d'un montant avec la méthode suivante :
`boolean makePurchase(long clientId, int amount)`
Ce service va lui-même contacter le service web de base de données pour d'abord vérifier le solde du client, s'il n'a pas les fonds nécessaires pour effectuer l'achat, la méthode renvoie faux et le client conserve son solde.
Sinon son solde est déduit du montant donné en paramètre et la méthode renvoie vrai.
Le scope de service web est à « **Application** ».

Base de données – Service Web

Pour simplifier le projet, les différents services accèdent à une unique base de données.
C'est une base **PostgreSQL** accessible depuis Java via l'API **JDBC**, les informations de connexion sont spécifiées par les constantes dans la classe `DataBase`.
Un dump de la structure de la base utilisée est fournie avec le projet.

Le service propose des méthodes pour accéder aux données et les mettre à jour.
Son scope est à « **Application** »

EiffelCorp – Service Web

C'est le service principal du projet, qui rassemble les autres.
Il permet à un client d'ajouter un ou plusieurs véhicules dans un panier (un panier par client donc ce service web a un scope à « **Session** ».

Ce service appelle les autres, il appelle par exemple le service :
- RMI IfsCar pour déterminer si un véhicule est disponible.
- Banque pour acheter un ou plusieurs véhicules.
- De convertisseur de devise pour obtenir un prix dans différentes devises.
- De base de données pour obtenir des informations sur un véhicule.

Interface web

L'interface proposée pour interagir avec ces services est issue d'une application web qui utilise la technique **JSP** (**J**ava**S**erver **P**ages) basée sur Java qui permet de créer dynamiquement du code HTML. Ceci nous permet notamment de faire appel aux services web (et RMI) pendant le chargement du code HTML et d'intégrer le résultat dans un contenu statique.

L'utilisation de cette application est détaillée dans le manuel utilisateur.

Installation et utilisation des services

Le projet se compose de 3 projets développés avec **Maven** (pour faciliter la gestion des dépendances) :
- Banque, Database et EiffelCorp. Ce dernier contient le service web IfsCarsService, le service RMI présenté précédemment, ainsi que le contenu de l'application web.
Ces 3 parties sont situées dans le même projet mais sont indépendantes.

Un dump de la base de données (structures + données) est fourni. Les informations de connexion sont en en-tête de la classe `fr.uge.database.DataBase` du projet `Database`.

Pour qu'un service web se connecte à une base de données il est nécessaire d'installer un driver JDBC sur le serveur. Cela se traduit ici par l'ajout dans le répertoire *lib/* de **Tomcat** d'un fichier *.jar*.
Dans ce projet nous utilisons une base postgreSQL et le jar en question se situe dans les dépendances Maven du projet `Database`.
Il est également disponible sur le repos Maven officiel à cette adresse.

Lorsque tout est installé et que les 3 projets sont ajoutés au serveur Tomcat, il faut le démarrer **puis** lancer la classe `fr.uge.ifsCars.Server` du projet `EiffelCorp` en tant qu'application Java pour démarrer le binding des objets RMI.
Si la connexion à la base de données a réussie il devrait apparaître dans les logs le message « Connected to database ! », sinon un message d'erreur s'affiche.

L'application web est disponible en local à l'adresse suivante :
http://localhost:*port*/EiffelCorp/
port = port utilisé par le serveur Tomcat (par défaut 8080).

Divers

Les sources de ce projet sont disponibles sur le git du projet.