TICILIA

Ameyes

M2 MIAGE FIC

|  |
| --- |
| **Rapport de projet**  **Application Java distribuée** |

Système de gestion de parc automobile & locations

Benoit charroux

[benoit.charroux@efrei.fr](mailto:benoit.charroux@efrei.fr)

2018-2019

Sommaire

Table des matières

[I. Fonctionnalités 3](#_Toc8616588)

[II. Utilisation 5](#_Toc8616589)

[III. Base de données : 8](#_Toc8616590)

[IV. Code : 9](#_Toc8616591)

[V. Conclusion : 9](#_Toc8616592)

**Table des illustrations**

[Figure 1 : Ajout d'une voiture 5](#_Toc8616665)

[Figure 2: Active MQ (Message) 5](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616666)

[Figure 3 : ActiveMQ 5](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616667)

[Figure 4: Résultat de l'ajout d'une voiture 6](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616668)

[Figure 5: Console H2 (URL : jdbc:h2:./src/main/resources/asnDB) 6](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616669)

[Figure 6 : Résultat ajout voiture coté base de donnés : 6](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616670)

[Figure 7 : Ajout d'un client - Database side 7](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616671)

[Figure 8 : Ajout d'un client 7](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616672)

[Figure 9: Enregistrement d'une location 7](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616673)

[Figure 10 : Enregistrement d'une location - database side 7](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616674)

[Figure 11: Structures des données/entités 8](file:///C:\Users\ameye\Desktop\CarRental\Rapport%20de%20projet%20JAVA.docx#_Toc8616675)

# Fonctionnalités

Voici une application java permettant la gestion des locations d’un parc automobile, et de ses clients. Cette application permettra d’ajouter de nouveaux véhicules à un parc automobile au sein d’une base de données, de modifier le prix des locations pour s’adapter à l’offre et la demande, ainsi qu’à louer une voiture en associant une voiture et un client.

#GET

/cars -> affiche toutes les voitures

#GET

/customers -> affiche tous les clients

#GET

/rentsbycar/{plateNumber} -> affiche toutes les locations enregistrées d’une voiture

#GET

/cars/{plateNumber} -> recherche d’une voiture par plaque d’immatriculation

#DELETE

/cars/{plaque} -> supprimer une voiture (par plaque d’immatriculation)

#DELETE

/customers/{idDriverLicense} -> supprimer un client (numéro de permis de conduire)

#PUT

/cars/prixjour/{plaqueVoiture} -> modifier le prix journalier de location d’une voiture

Body : nouveau prix

#POST

/cars/add -> ajouter une voiture

Body : voiture (format JSON)

{

"plaque" : "plaque d’immatricution",

"nombrePlaces" : X,

"dailyPrice" : prix journalier (nombre),

"caution" : prix de la caution (nombre),

"marque" : "Marque de la voiture",

"modele" : "Modèle de la voiture",

"type" : "type de véhicule (sportive, break, …)"

}

#POST

/customers/add -> ajouter un client

Body : client (format JSON)

{

"idDriverLicense" : "numero de permis",

"firstName" : "Prenom",

"lastName" : "Nom de famille",

"nationality" : "Nationalité"

}

#POST

/rents/add -> ajouter une location

Body : client (format JSON)

{

"plaque": "plaque d’immatriculation",

"idDriverLicense": "numéro de permis",

"dateDebut" : "ANNEE-MM-DD",

"dateFin": "ANNEE-MM-DD",

}

# Utilisation

Pour utiliser l’application, vous devez passer par une interface plugin de navigateur web. J’ai utilisé pour ce faire le plugin postman qui permet l’envoie de requêtes paramétrées et affiche les réponses serveur.

Initialement, la base de données que l’on étudiera par la suite est vide. Vous devez alors commencer par alimenter la base.

Ouvrez votre invite de commandes, placez vous dans le répertoire du dossier décompressé et lancez les commandes gradlew build puis gradlew eclipse.

Ouvrez ensuite votre IDE favori et importez le projet. Exécutez alors le programme en lançant CarRentalApplication.

Ouvrez une seconde invite de commande et lancez activeMQ depuis le dossier bin de son répertoire (activemq start)

Commençons par ajouter une voiture et un client :

N’oubliez pas de configurer votre body au fomat JSON.

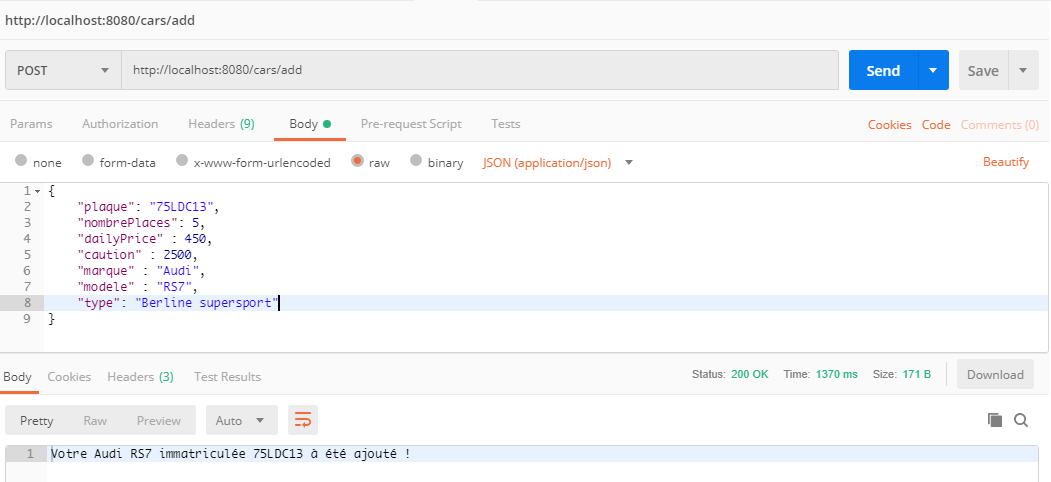
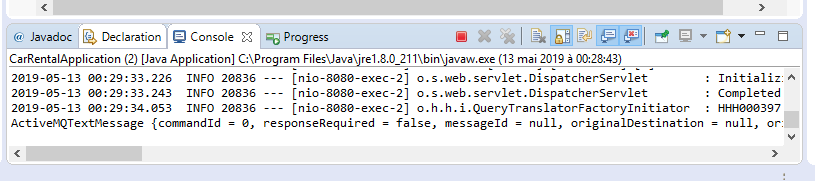


Figure : Ajout d'une voiture

La voiture est ajoutée à la base ! Vous verrez et ce à chaque requête, d’une part une réponse serveur confirmant l’action établie ainsi qu’un message broker reçu via la console de votre IDE.

N’hésitez pas à

Zoomer si nécessaire

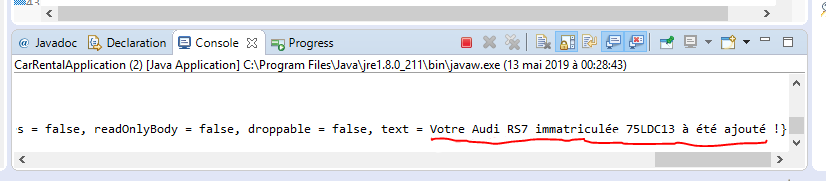


Figure : Active MQ (Message)

Figure : ActiveMQ

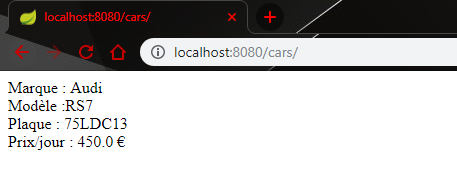
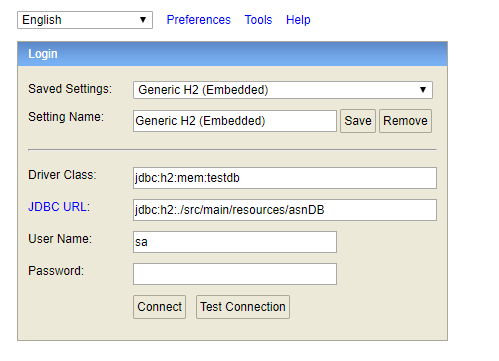
Voyons le résultat :

Figure : Résultat de l'ajout d'une voiture

L’ajout d’une voiture dont la plaque est déjà présente dans la base sera bloquée.

Voyons maintenant notre base de données. Connectez vous à la console H2, très pratique, qui permet la gestion de sa base de données légère et fonctionnelle in-memory. URL :

Veuillez cependant

Respecter l’URL indiquée

Figure : Console H2 (URL : jdbc:h2:./src/main/resources/asnDB)

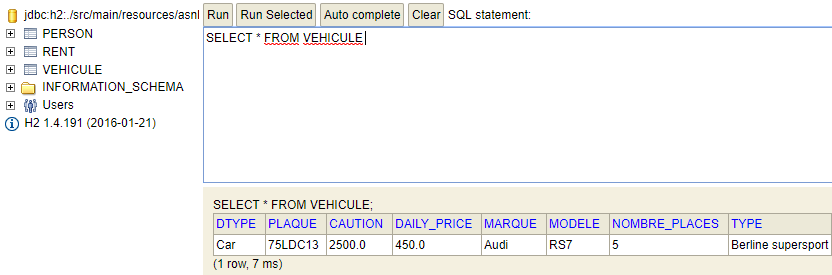
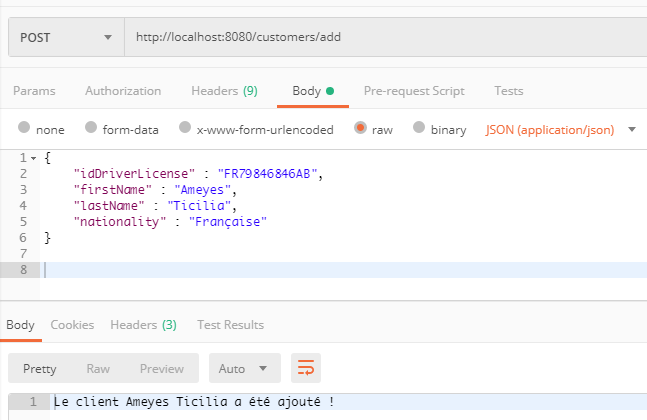


Figure : Résultat ajout voiture coté base de donnés :

L’ajout d’un client est assez similaire :



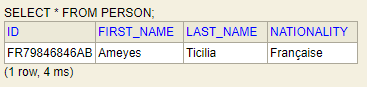


Figure : Ajout d'un client - Database side

Figure : Ajout d'un client

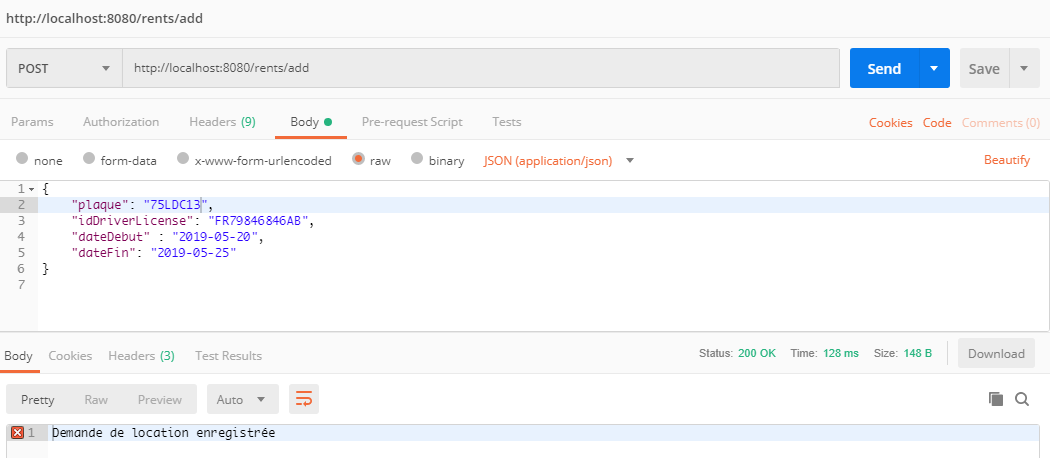
Essayons à présent de louer cette voiture pour une date donnée :

Figure : Enregistrement d'une location



Figure : Enregistrement d'une location - database side

Le prix de la location est automatiquement calculé en fonction de la période saisie et du prix journalier. Les fonctionnalités et requêtes restantes mise à part l’interface client sont assez similaires, nous ne les détaillerons donc pas ici.

# Base de données :

L’application possède une base de données relationnelle de trois tables. Illustrons son fonctionnement grâce à un diagramme de classes des entités JPA associés :

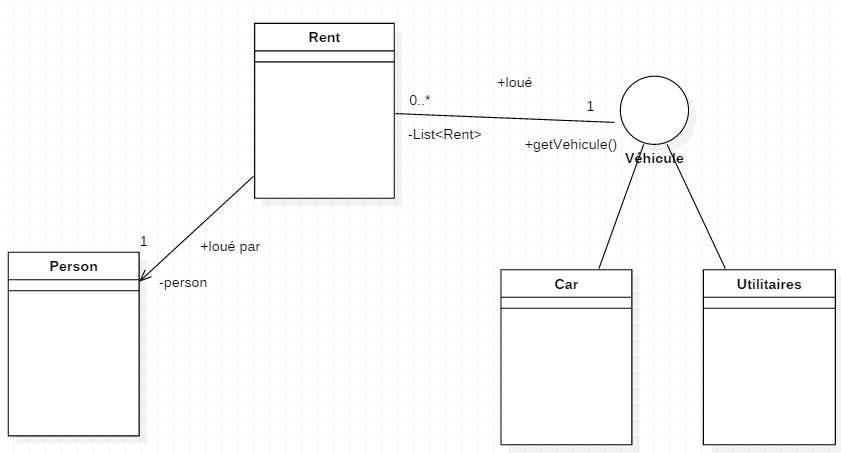


Figure : Structures des données/entités

L’interface véhicule nous permet de centraliser la gestion des véhicules. Nous pourrons par la suite développer toutes sortes de véhicules et leurs spécificités et les intégrer directement au système. Un véhicule possède une liste de locations enregistrés (en cours, futures ou passés). Nous pourrons alors vérifier à l’aide d’une fonction (isAvailable dans le code) la disponibilité de chaque voiture sur une période donné. Un objet rent est quant à lui unique à chaque location. Il possède l’ensemble de l’objet voiture associé et de l’objet person correspondant au loueur. Il est donc possible depuis un objet rent de remonter les informations à la fois du conducteur et du loueur. Un objet rent par définition doit être associé à un seul et unique trinôme loueur-voiture-date, et ne peut exister sans la présence d’un des trois. Dû à quelques problèmes techniques de jointures, je n’ai pas eu le temps de finaliser la liaison de l’entité person au reste des entités.

# Code :

Ce programme contient onze classes. On y retrouve les classes modélisant les entités (Car, Rent, Person, Utilitaire) et leur interface (Véhicule).

On trouve également deux interfaces CarRepository et PersonRepository, permettant l’utilisation de fonctions spring tels que findByXXX(Valeur) qui permet de retrouver un objet parmi une liste d’objets du même type selon l’attribut XXX et ceux, grâce à l’extension de l’interface CrudRepository.

On y trouve également deux fichiers de services. LocationServices contient le controleur de l’application et sert de routeur au différentes URL mentionnées plus tôt, et de message broker. On y trouve également le code des traitements fonctionnel appelés via ces URLs ce qui en fait un « contrôleur-modèle ».

# Conclusion :

Ce projet fut l’occasion pour moi, appréciant particulièrement le développement d’application JAVA, l’occasion d’apprendre à développer des applications web distribuées très en vogue depuis quelques années à l’aide du framework spring, de la base de données H2 qui permet un déploiement une exécution et des tests beaucoup plus rapides. Je me suis ainsi également familiarisé avec les protocoles de messagerie via les messages brokers qui permettent l’envoi de messages entre applications. Ces notions m’apporterons probablement beaucoup par la suite étant par ailleurs actuellement en contrat de professionnalisation chez BNP Paribas au sein de l’entité global market, filiale boursière du groupe où le développement d’applications JAVA de type similaires sont en oeuvre.

Ce projet constitue ainsi une base, relativement correcte, mais devant être approfondie dans l’optique de maitriser tous les atouts des technologies utilisés. On pourra alors développer un système beaucoup plus complet comprenant la gestion correcte des profils clients, des paiements ou encore de compléter les types de véhicules pris en charge. L’interface devra également être améliorée où plutôt crée pour communiquer avec le contrôleur afin de ne plus avoir à utiliser de plugin pour navigateur. J’ai voulu durant ce projet concentrer mes recherches et travaux sur l’aspect serveur de l’application afin d’étudier au mieux les fonctionnalités du framework spring, des concepts de JPA pour la persistance où encore de message broker. L’interface m’apportant selon moi moins de valeur ajoutée en compétences dans la mesure où le principe reste le même pour n’importe quelle application.