Rapport de projet

ComicarsRental – API de Gestion de Location de Véhicules

Alessio Comi

He-Arc Neuchâtel	Janvier 2024
Classe	ISC3-il-a
Unité d'enseignement	3291.2 JEE/Spring 1
Enseignant et assistant	M. Chèvre Sébastien

1 INTRODUCTION

ComicarsRental constitue une API dédiée à la gestion de la location de véhicules, spécialement imaginé pour simplifier le processus de location de voitures au sein du garage Comicars. Cette API propose une plateforme centralisée qui permet aux utilisateurs de rechercher, réserver et administrer les véhicules de manière pratique et conviviale.

2 ARCHITECTURE IMPLÉMENTÉE

L'architecture de l'application ComicarsRental repose sur un modèle de micro-services, chacun d'entre eux étant responsable d'une partie spécifique de la fonctionnalité globale. Trois principaux micro-services ont été mis en place : « Vehicle », « User », et « Reservation ».

Le premier gère les informations relatives aux véhicules disponibles à la location. Il expose des fonctionnalités telles que la récupération de tous les véhicules, la recherche de véhicules en fonction de critères spécifiques tels que la marque, le modèle, l'année de fabrication et le taux de location, ainsi que la possibilité d'ajouter, mettre à jour et supprimer un véhicule. L'accès à ces fonctionnalités est fourni par le biais du service « VehicleService », qui agit comme une interface entre le micro-service et le reste de l'application.

Le deuxième gère les informations sur les utilisateurs de l'application. Il offre des fonctionnalités telles que la récupération de tous les utilisateurs, la recherche d'utilisateurs par identifiant, la création, la mise à jour et la suppression d'utilisateurs. Le service associé, « UserService », gère les interactions avec ce micro-service.

Le troisième est dédié à la gestion des réservations de véhicules. Il permet de récupérer toutes les réservations, de rechercher des réservations par utilisateur, par véhicule, par plage de dates et par statut. De plus, il offre la possibilité de créer et d'annuler une réservation. Le service associé, « ReservationService », interagit avec ce micro-service pour fournir ces fonctionnalités.

L'ensemble de ces micro-services utilise des repositories pour accéder aux données stockées dans des bases de données spécifiques à chaque entité.

En termes d'API, chacun d'entre eux expose des points d'accès clairement définis, au travers des contrôleurs REST, respectant les principes CRUD pour les entités associées. Les services agissent en tant qu'interfaces entre les contrôleurs et la logique métier du micro-service, garantissant ainsi une communication standardisée et modulaire.

3 CONCEPTION

3.1 INTRODUCTION

Initialement, l'API a été conçue pour compléter un projet personnel appelé « ComicarsManagement ». Pour ce faire, l'authentification des utilisateurs devait passer par une classe appelée « SecurityConfig », utilisant des jetons JSON Web Token (JWT) fournis par un service de type provider. De plus, un package appelé « payload » était initialement envisagé avec deux rôles distincts.

Premièrement, ce package devait s'occuper de l'encapsulation des informations client une fois celui-ci authentifié. Il devait également fournir un « tokenType Bearer », ultérieurement utilisé dans une double authentification de type OAuth 2.0.

Deuxièmement, le package « payload » était destiné à gérer la sérialisation et la désérialisation des données à travers des objets de transfert de données (DTO).

3.2 PROBLÈME

Rapidement, l'authentification des utilisateurs au moyen des jetons s'est révélée être drastiquement plus complexe qu'envisagé. Les défis rencontrés ont généré des implications sur la planification initiale du projet, induisant ainsi un dépassement des délais impartis. La complexité inattendue de la mise en œuvre de cette fonctionnalité a conduit à une réévaluation des priorités de développement, soulignant la nécessité de repenser l'approche initiale.

De plus, la logique de gestion des réservations s'est avérée être plus complexe que prévu. En raison du premier problème rencontré avec l'authentification des utilisateurs, la réévaluation des priorités n'a pas pu être menée de manière réfléchie, entraînant des retards supplémentaires dans le processus de développement. Cette situation a accentué les défis initiaux, nécessitant une réflexion approfondie sur la manière d'aborder simultanément les deux aspects critiques du projet tout en maintenant la qualité et l'efficacité globales.

3.3 RÉSOLUTION

Pour faire face au problème rencontré, une révision des fonctionnalités utilisateur a dû être réaliser, de fait la portée a été temporairement restreinte pour se concentrer sur la création d'utilisateurs sans implémentation de l'authentification. Bien que l'attribution des rôles ne soit pas actuellement sécurisée, cette décision ne pose pas de problème majeur dans le contexte de l'API. Cette approche permet de prioriser le développement des fonctionnalités essentielles, avec l'intention d'ajouter des couches de sécurité telles que l'authentification dans les itérations futures du projet.

3.4 CHOIX

Le choix de mettre en œuvre une architecture de micro-services a été motivé par la volonté d'assurer une modularité et une évolutivité optimales. De plus, les technologies utilisées, telles que les repositories pour accéder aux données et les contrôleurs REST pour exposer les points d'accès, ont été sélectionnées pour garantir une communication standardisée et s'inscrire dans la ligné des consignes. Ces choix stratégiques visent à créer une base solide pour l'évolutivité future de l'API tout en assurant une cohérence dans son fonctionnement global et respectant les conventions et bonnes pratiques.

3.5 TEST

Uniquement les tests liés aux fonctionnalités principales de l'API ont été implémentés. Ceux-ci se concentrent sur les parties fonctionnelles clés de l'application et couvrent des aspects tels que la vérification de l'existence d'utilisateurs par ID, la recherche d'utilisateurs par nom d'utilisateur, la sauvegarde d'utilisateurs et la récupération de la liste complète des utilisateurs. Ces tests sont conçus pour garantir le bon fonctionnement des fonctionnalités centrales de l'API, mais d'autres test sont nécessaires pour augmenter la couverture du code.

4 PLANNING

Si le planning initial se voulait segmenter avec pour objectif d'atteindre régulièrement des Milestones, la réalité s'est révélée être quelque peu différente. En effet, la masse de travail ininterrompue provenant des autres unités d'enseignement, comportant des échéances antérieures à celles de ce projet, a eu un impact significatif sur l'organisation. Cette charge de travail imprévue a nécessité une redistribution des ressources et a entraîné des ajustements dans la gestion du temps.

Cependant, étant donné que les délais impartis ne pouvaient pas être étendus, une adaptation des fonctionnalités a été entreprise pour répondre aux contraintes temporelles. Cette approche a exigé une réévaluation minutieuse des priorités et une révision du périmètre initial du projet. Certains aspects ont été simplifiés, tandis que d'autres ont été abandonnés.

L'implication directe de ces contraintes a été la nécessité de prendre des décisions stratégiques pour maximiser l'efficacité et l'impact du projet dans les limites du calendrier imposé. Cette situation souligne l'importance de l'organisation proactive des étudiants, afin de pouvoir être en mesure de réaliser ces objectifs dans les délais imparties.

5 BILAN

Malgré des avancées significatives, seule une partie de l'API est parfaitement fonctionnelle, couvrant la gestion des véhicules ainsi que la gestion simplifiée des utilisateurs. Cependant, la gestion des réservations a dû être momentanément mise de côté en raison de contraintes imprévues.

La première étape de la suite du projet consistera à reprendre la logique des réservations et à finaliser son implémentation. Une fois cette phase achevée, l'attention se portera sur l'authentification, en s'alignant sur les exigences du cahier des charges, ainsi que sur le développement des fonctionnalités secondaire. Enfin, la dernière phase du projet consistera à créer une interface avec Thymeleaf afin de fusionner harmonieusement avec ComicarsManagement. Ces étapes successives visent à compléter l'API, renforçant ainsi sa fonctionnalité et son intégration globale dans le contexte du projet.

6 GUIDE D'INSTALLATION

Avant de commencer, assurez-vous d'avoir Git, Maven et Postman installé sur votre machine. Ensuite vous pourrez utiliser ces trois commandes pour lancer l'application

- git pull
- mvn clean install
- mvn spring-boot:run

Par la suite, ouvrez Postman, puis importez la collection comicarsRental.postman_collection.json en utilisant l'option "Import" dans le coin supérieur gauche. Vous êtes maintenant prêt à explorer et tester les API de ComicarsRental à l'aide de Postman. Assurez-vous que l'application est en cours d'exécution avant d'envoyer des requêtes.