Le génie pour l'industrie

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS LOG792

CAN YOU DRIVE ME?
UNE SOLUTION AU COVOITURAGE

DAVID FRANCOEUR-RAYMOND
FRAD04109006

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

ALAIN APRIL

MONTRÉAL, 05 AOÛT 2015
ÉTÉ 2015

CAN YOU DRIVE ME?
UNE SOLUTION AU COVOITURAGE

DAVID FRANCOEUR-RAYMOND

RÉSUMÉ

Ce rapport présente la conception d'un système d'arrière-plan de covoiturage. Ce système à partir d'un document de spécifications des requis logiciels (53). Ce rapport est un document technique dans lequel les différentes décisions de conception et d'implémentation qui ont été prises afin de satisfaire ces exigences sont détaillées.

Le système est conçu en deux parties importantes : un système d'arrière-plan (backend) et un système de gestion de données.

Le système d'arrière-plan est une application offrant différents APIs avec lesquels les utilisateurs peuvent récupérer, ajouter, modifier ou supprimer des données. Ces APIs sont accessibles à l'aide de requête HTTP (RESTful API).

Ce rapport ne vise pas à définir comment les différents clients qui utiliseraient ce système d'arrière-plan pourraient être conçus. La distinction entre ce système et les différents clients est particulièrement importante puisqu'une fois que ce système est stable, il devient simple d'ajouter d'autres clients : application mobile (iOS, Android,Windows Phone, etc.) ou application de bureau (Windows, Mac, Linux, etc.). Il est aussi simple d'intégrer le service avec d'autres services externes.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

SGV	Système de gestion de version
SIC	Système d'intégration continue
JVM	Java virtual machine
SMT	Système multitenanciers
SGS	Système de gestion de schéma
API	Application programmable interface
BDD	Base de données
SGBD	Système de gestion de base de données
JWT	Json Web Token

INTRODUCTION

Problématique et contexte

Les gens qui se déplacent fréquemment de région en région devraient pouvoir compter sur une solution simple, efficace et économique afin d'organiser leurs déplacements. Malheureusement, les services d'autobus ou de trains sont souvent coûteux et n'offrent pas d'horaires très variés. Il existe donc un réel besoin pour une solution de covoiturage économique au Québec.

Le problème est simple et peut être défini selon deux perspectives : celle du passager et celle du conducteur. Pour le passager, il s'agit de trouver un transport d'un point A à un point B à une date et une heure précise. Pour le conducteur, il s'agit de combler les places restantes dans son véhicule afin de rentabiliser ces déplacements.

Il existe actuellement des solutions, mais elles ne sont pas adaptées à la clientèle de ce genre de service. Les solutions existantes n'offrent pas un bon rapport qualité-prix. On peut observer le mécontentement de la clientèle avec l'apparition de groupe Facebook pour combler le manque de solution efficace.

Ce projet vise donc à entreprendre la conception d'un service de covoiturage simple, efficace et très peu coûteux qui permettra aux gens de facilement planifier leurs déplacements de longue distance.

CHAPITRE 1

OBJECTIFS ET MÉTHODOLOGIE

1.1 Objectifs du projet

Le projet vise à produire un système qui apportera une solution fiable, économe et rapide aux problèmes de covoiturage. Pour y arriver, le projet définit les objectifs suivants :

- Fournir un système d'arrière-plan (backend) avec API REST

Ce système permettra à différentes interfaces client de fonctionner correctement.

1.1.1 Système d'arrière-plan indépendant

Ce système sera responsable de traiter les demandes des différentes interfaces qui pourraient éventuellement être conçues : Application web, Android, iOS, etc.

L'indépendance du système est définie ainsi : capacité du système à opérer indépendamment des autres composantes du projet. Dans le cadre du projet, cela signifie que l'avancement, le déploiement, la maintenance du système d'arrière-plan ne sont en aucun cas rattachés au cycle de vie des différentes interfaces utilisateurs.

Évidemment, si les diverses interfaces ont besoin de fonctionnalités qui ne sont pas encore offertes par le système d'arrière-plan, ce dernier devra être adapté afin de répondre aux besoins. Cependant, il serait tout à fait normal que le système d'arrière-plan offre un ensemble de fonctionnalités qui ne sont pas toutes utilisées par les diverses interfaces.

1.2 Hypothèses

Ce projet vise à concevoir un système d'arrière-plan offrant une solution de covoiturage accessible via un navigateur. Cependant, il existe plusieurs outils qui permettent de réaliser ce genre de système rapidement avec un minimum d'effort.

Afin de maximiser l'effort de conception, les hypothèses suivantes sont émises et devront être prises en compte lors de la conception du système :

- le déploiement du système devra être complètement automatisé
- des tests d'intégrations devront pouvoir s'exécuter dans différents environnements
- le système sera multitenanciers

Les hypothèses sont décrites plus bas et les approches utilisées pour concevoir un système les respectant sont définies dans les chapitres suivants.

1.2.1 Déploiement automatisé

Le déploiement automatisé est une propriété intéressante d'un système. Aussi appelé « Déploiement continu » (Continuous Delivery), le déploiement automatisé permet la livraison rapide de nouvelles fonctionnalités ou de correctifs.

Le déploiement automatisé consiste en l'automatisation des tâches qui sont requises pour effectuer la livraison d'un changement dans le système. Le déploiement automatisé n'exige pas nécessairement que les changements soit automatiquement appliqués en production, cependant, les changements doivent être prêt à être manuellement déployé dans cet environnement le cas échéant.

1.2.2 Tests d'intégrations multienvironnements

Le système devra être testable dans tous les environnements où il peut opérer à l'exception de la production. Parmi les différents environnements dans lequel le système peut opérer, on retrouve :

- développement : sur le poste du développeur
- ci : sur le système de déploiement automatisé
- qa: environnement ayant les mêmes propriétés que la production (infrastructures et données)

- production : environnement où le système est en mode opérationnel

Il peut exister d'autres environnements. Dans le cadre de ce projet, il n'existera que deux environnements : développement et production.

1.2.3 Système multitenanciers

Le système devra être conçu afin d'être multitenanciers (multi-tenants). Dans une application web habituelle, un seul schéma de base de données est requis pour tous les clients. Dans un système multitenanciers, chaque client possède sa propre base de données.

Ainsi, les données du client A ne sont pas dans la même source de données que ceux du client B. Cela permet d'éviter que l'information de deux tenanciers puisse être mélangée.

CHAPITRE 2

SYSTÈME DE SOLUTION AU COVOITURAGE

Afin de concevoir un système d'arrière-plan moderne, plusieurs éléments doivent être pris en compte. Ce chapitre abordera ces éléments :

- Choix reliés aux technologies utilisées
- Les techniques de tests
- La sécurité

2.1 Technologies

Un système d'arrière-plan exposant un API (Application Programmable Interface) RESTful est un système complexe qui réalise plusieurs tâches. Afin de réaliser le développement d'un tel système, une série d'outils, de langages, etc. devront être sélectionnés.

Cette section indiquera les choix technologiques qui ont été faits dans le cadre de ce projet.

2.1.1 Technologie du système de gestion de version

Le système de gestion de version (SGV) est un outil essentiel de tout projet logiciel. Cet outil permet de conserver un historique des travaux effectués (*voir* Système de gestion de version). La plupart des artéfacts produits au cours du développement logiciel sont conservés dans cet outil. De plus, le SGV permet de synchroniser le travail de plusieurs personnes facilement.

2.1.1.1 Technologies disponibles

Il existe plusieurs outils différents afin de gérer les versions des artéfacts produits au cours de la conception et de la réalisation d'un projet logiciel. Voici une liste non exhaustive des alternatives disponibles :

- Subversion
- Mercurial
- Git
- CVS

Les différentes options présentées ont toutes leurs avantages et désavantages. Cependant, dans le cadre du projet, il est peu utile de comparer ces différents outils.

2.1.1.2 Choix de la technologie

Le système de gestion de version qui sera utilisé est Git. Ce système est simple d'apprentissage, efficace et distribué. De plus, l'expérience de l'ingénieur avec Git est non négligeable.

Il existe plusieurs services qui offrent l'hébergement de « repository » Git privé et gratuit. Git est également le système de gestion de version le plus populaire actuellement. Sa présence dans pratiquement tous les projets est incomparable. Git est aussi facilement adaptable à différentes techniques de travail (workflow). De plus, plusieurs outils permettant la gestion de déploiement automatisé s'intègrent avec Git.

2.1.2 Technologie du système d'intégration continue

Le système d'intégration continue (SIC) permet d'automatiser les différentes étapes du déploiement logiciel (*voir* Système d'intégration continue). Ces outils sont souvent des systèmes complexes à part entière qui servent à accélérer et standardiser le processus de livraison.

2.1.2.1 Technologies disponibles

Il existe plusieurs systèmes qui permettent de gérer l'intégration continue d'un système. Le plus mature et le plus populaire étant probablement Jenkins. Cependant, voici quelques alternatives qui peuvent être considérées :

- Travis CI

- Circle CI
- Bamboo

Plusieurs de ces alternatives sont intéressantes, cependant, la plupart sont coûteuses. Jenkins est un logiciel open source très populaire et très versatile, cependant, l'installation, la configuration et la maintenance d'un tel système sont exigeant.

2.1.2.2 Choix de la technologie

Dans un souci de simplicité, Jenkins a été choisi. Bien qu'il soit laborieux à configurer et à installer, il est gratuit et open-source et de plus, permettra au processus de déploiement de travailler avec Docker.

Jenkins pourra compiler et tester le code et ensuite le déployé sur la plateforme déterminée.

2.1.3 Technologies pour l'implémentation du système d'arrière-plan

Dans le cadre de ce projet, le système consiste en un API accessible via HTTP. Ce système devra agir en tant que serveur web. Il devra traiter les requêtes des clients en récupérant l'information ou en écrivant de l'information dans une base de données. Ce système devra aussi être hautement testable.

2.1.3.1 Technologies disponibles

Pour réaliser un système d'arrière-plan qui répond aux exigences du projet, il existe de nombreuses alternatives qui pourraient être utilisées. L'utilisation de l'une ou de l'autre de ces technologies aura un impact important sur la réalisation.

Pour faire un choix éclairé, il serait judicieux de dresser une liste des langages et frameworks qui pourraient être utilisés afin de réaliser ce type système. La liste suivante présente donc une série de technologies qui sont familières à l'ingénieur avec une description des avantages, des inconvénients et des outils disponibles.

2.1.3.1.1 JavaScript

Depuis quelques années maintenant, il existe une recrudescence impressionnante entourant JavaScript. Avec l'arrivée de Node.js, une communauté aux bases très solides s'est formée. À l'origine, JavaScript était utilisé comme langage d'exécution du côté client (navigateur web). Node.js a permis l'utilisation de JavaScript comme un langage utilisé du côté serveur. C'est un avantage considérable de pouvoir développer le client et le serveur d'un système dans le même langage.

De plus, il existe plusieurs outils mis à la disposition de l'ingénieur pour rapidement mettre sur pied un prototype fonctionnel, élégant et pratiquement prêt pour la production. Voici une liste non exhaustive de ces outils :

- Yeoman – Permet de générer un prototype très rapidement. Plusieurs générateurs différents peuvent être utilisés afin de démarrer plusieurs types de projets. Les différents générateurs sont créés par la communauté et implémentent les meilleures pratiques de l'industrie.
- Bower – Permet de gérer les différents frameworks et librairies qui pourraient être nécessaires afin de réaliser un système web. Bower est optimisé particulièrement pour les dépendances de système d'avant-plan (frontend).
- Grunt – Permet de définir des tâches à effectuer dans le cadre du projet. Grunt, un peu à l'image de Maven, permet de gérer des tâches comme la compilation, les tests, le paquetage et le déploiement. Il est trivial de définir des tâches complètement personnalisées avec Grunt, alors que cela peut-être plus complexe avec Maven.
- npm – Permet de gérer les dépendances de son projet. Ici aussi, cet outil peut être comparé à Maven. Cependant, npm utilise une tout autre approche pour gérer les versions. La gestion de version de dépendances peut rapidement devenir complexe avec Maven. npm définit une version pour chaque dépendance et leurs sous-dépendances. Si une dépendance requiert une dépendance qui se trouve déjà dans l'arbre de dépendances, le conflit de version qui pourrait survenir sera évité, car npm inclura les deux versions de cette même dépendance.

Avec ces outils, JavaScript connait un véritable regain de vie et permet à des milliers de développeurs de rapidement démarrer des projets de qualité. C'est pourquoi il a été considéré dans le cadre de ce projet.

2.1.3.1.2 Meteor

Meteor, écrit en JavaScript, est une plateforme complète permettant le développement d'application web. Meteor est traité ici comme un langage différent puisque son utilisation implique l'utilisation de JavaScript dans l'ensemble du système et de MongoDB comme système de stockage de données.

Meteor a donc principalement les mêmes avantages que le JavaScript, indiqué plus haut, cependant, il fournit un ensemble de fonctionnalités intéressantes qui permettent la création de projets extrêmement rapidement. Meteor permet à la base de créer un site web, par contre, il est facile d'intégrer Meteor avec iOS et Android afin de créer de superbes applications mobiles. Les projets basés sur Meteor sont temps réel puisque les données sont mises à jour dans toutes les couches du système simultanément. C'est ici la force de Meteor : même si plusieurs éléments sont exigés par la plateforme, celle-ci permet le support de différentes plateformes facilement rapidement. Une application Meteor offre à la base des fonctionnalités qui sont extrêmement complexes à développer.

2.1.3.1.3 Java

Java est un langage qui célèbre ses 20 ans cette année. Java est mature et a su évoluer à travers le temps de façon remarquable. Ce langage de haut niveau permet d'être compilé et ensuite exécuté sur n'importe quelle plateforme où la Java Virtual Machine (JVM) est disponible.

La maturité du langage lui offre des propriétés intéressantes : stabilité, éprouvée en entreprise, documentée, etc. De plus, Java, comme JavaScript (et Node.js), offre une multitude d'outils, de framework et de librairies qui permettent de démarrer un projet de

qualité professionnelle rapidement. Voici, ici aussi, une liste non exhaustive d'outils qui font de Java est un langage de choix dans la réalisation de système d'arrière-plan :

- Spring – Permet de construire des projets de qualité professionnelle en utilisant des outils, des architectures et des patrons de programmation éprouvés en entreprise. Spring est très mature, bien documenté et entouré d'une communauté vibrante qui s'assure que le framework demeure à la fine pointe de la technologie.
- Hibernate – Permet de gérer tout l'aspect de persistance de données qui est nécessaire dans une application Java. Hibernate est, comme Spring, un framework mature et bien documenté qui est utilisé en entreprise. Hibernate permet la persistance dans un très grand nombre de systèmes de stockages de données différents, à la fois relationnels (SQL) et NoSQL. Son intégration avec Spring est triviale.
- Maven – Permet de gérer les dépendances, les tâches, les versions, le déploiement et bien d'autres choses. Maven est l'outil par excellence de tout projet Java. Maven attaque de front une panoplie de problèmes différents entourant la conception de système. C'est un outil polyvalent qui a lui aussi été éprouvé en entreprise.
- Gradle – Permet de gérer les dépendances, les tâches, les versions, le déploiement et bien d'autres choses. Gradle est un compétiteur direct de Maven. Cependant, Gradle est fondamentalement différent par son langage de définition (XML pour Maven). La définition de tâches personnalisées dans Gradle est simple.

Java est lui aussi un langage de choix dans la conception de système d'arrière-plan. De plus, l'ingénieur possède une grande expérience avec Java.

2.1.3.2 Choix de la technologie

Dans le cadre de ce projet, Java a été retenu comme technologie qui sera utilisée pour réaliser l'implémentation système d'arrière-plan. Le choix a été fait en prenant en considération l'expérience de l'ingénieur, la maturité de la technologie et la documentation disponible pour cette technologie. À l'aide de Spring Boot (une des

composantes de Spring), le projet sera rapidement sur pied et pourra tirer avantage d'Hibernate. Cette combinaison permettra un maximum de liberté quant au choix du système de gestion de base de données.

Gradle sera utilisé pour gérer l'ensemble des tâches entourant le code source du projet : compilation, tests, déploiement, etc. Gradle a été préféré à Maven par sa flexibilité et sa gestion de dépendance améliorée.

2.1.4 Technologies pour le système de gestion de base de données

Le système de gestion de base de données (SGBD) est une partie essentielle du système d'arrière-plan. Il permet la sauvegarde, la mise à jour et la récupération de toutes les données qui donnent un sens au système. Il est donc important d'utiliser une technologie appropriée aux besoins, mature et performante.

2.1.4.1 Type de technologies

Dans le monde du stockage de données, ce ne sont pas les options qui manquent. En réalité, il existe tellement de produits différents sur le marché qu'il peut être laborieux de déterminer la bonne technologie à utiliser. En théorie, chacune de ces alternatives vise une utilisation précise et il s'agit de définir ces besoins avec précision afin de trouver la solution optimale.

Dans le monde du stockage de données, il existe plusieurs types de technologies différentes :

1. relationnel (SQL)
2. orienté objet
3. non relationnel (NoSQL)
    a. graph
    b. clé-valeur
    c. documents
    d. famille de colonnes

4. etc.

Ces différentes technologies offrent des solutions à des problèmes différents. Le but de cette section n'est pas de comparer ni d'expliquer les différences entre ces technologies, mais d'indiquer quelle technologie sera utilisée et d'expliquer ce choix.

Dans le cadre de ce projet, les requis quant au stockage de données sont assez relâchés. À partir des différents cas d'utilisation, il est simple de faire une esquisse des liens entre les différentes entités du système. Veuillez noter que ce diagramme n'est qu'une esquisse des relations interentités (*voir* Conception du schéma de données)

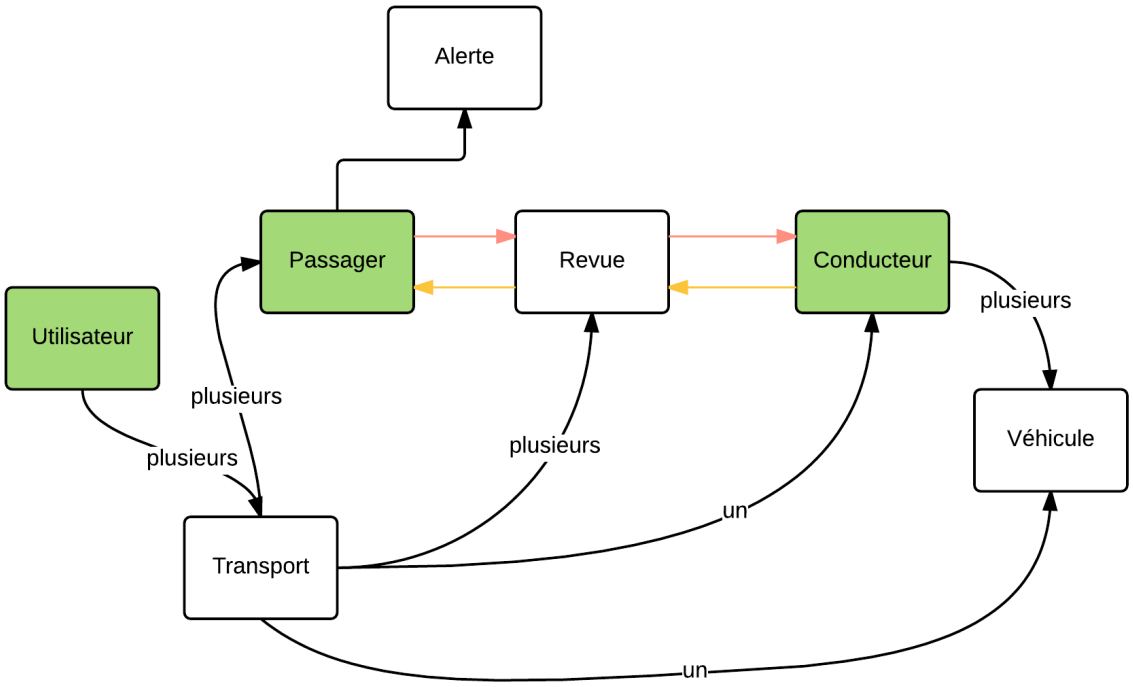

Figure 2-1 - Relation interentités de haut niveau

La figure ci-dessus montre une vue de haut niveau des relations entre les différentes entités du système. Cette vue permet de voir qu'il existe une multitude de relations entre les entités. Le système serait donc à première vue un excellent candidat pour un système de base de données basé sur un modèle relationnel.

Par contre, une observation intéressante peut être tirée de ce diagramme. Il peut devenir laborieux de récupérer toute l'information rattachée à un utilisateur. Une telle requête nécessite l'utilisation de plusieurs jointures.

Il pourrait être intéressant de considérer un système de base de données utilisant un modèle non relationnel orienté vers le stockage de documents. Dans ce genre de système, une seule requête, sans jointure, permettrait de récupérer l'ensemble des informations reliées à un utilisateur. Le document utilisateur contiendrait ainsi l'ensemble des informations : véhicules, transports, revues, etc.

Dans le diagramme, les bulles vertes montrent les entités qui sont d'un même type. Ici, l'utilisateur, les passagers et le conducteur d'un transport représentent tous le même type de données. Par conséquent, lors de la sauvegarde d'un tel document, il y aura de la duplication de données ce qui rendra la mise à jour de ces données plus difficile.

Ce type de redondance peut être lourd à gérer et c'est pourquoi le type de SGBD utilisé sera : relationnel. Les systèmes de gestion de base de données relationnelle sont populaires et présents en industrie depuis plusieurs années.

Deux candidats ont été retenus :

1. MySQL
2. PostgreSQL

Ces candidats ont été retenus pour plusieurs raisons :

1. Maturité
2. Documentation
3. Disponibilité de framework (ORM) dans plusieurs langages

2.1.4.2 Choix de la technologie

Le candidat retenu pour ce projet sera : PostgreSQL. Ce choix aura été basé sur la nature « ouverte » (Open source) du système puisqu'en réalité, il est très probable que ces deux systèmes offrent des avantages et désavantages similaires pour un tel projet.

Bien que l'ingénieur ait une plus grande expérience avec MySQL, ce projet, qui est à la base académique, servira d'apprentissage pour l'ingénieur.

2.2 Techniques de tests

Les tests sont une partie intégrante du développement logiciel. Ils ne doivent donc pas être négligés. Pour assurer la qualité du logiciel, les tests doivent être implémentés, maintenus et le résultat de leurs exécutions doit être pris en compte.

Les choix technologiques décrits plus haut guideront une partie de la stratégie de test. Les outils et langages utilisés afin d'implémenter l'application sont exhaustivement testés et offrent une multitude de fonctionnalités permettant de tester un système.

Le système d'arrière-plan, écrit en Java, avec l'aide de Spring et Hibernate, utilise Gradle en tant que gestionnaire de source. Gradle permet la définition de tâches et le plug-in « java » ajoute les tâches suivantes au projet :

Parmi ces tâches, on retrouve :

- clean : supprime l'ensemble du contenu généré (supprime le dossier build/)
- build : compile et test le projet
- jar : construit un fichier JAR pour permettre la distribution du projet
- javadoc : génère la documentation au format JavaDoc à partir des sources
- test : exécute les tests
- dependencies : affiche un arbre des dépendances de l'application (très utile)

Ici, la tâche « test » est celle qui sera utilisée afin d'exécuter les tests. Cette tâche ne fait aucune distinction entre les tests unitaires et les tests d'intégration. Cependant, ces deux types de tests sont différents.

Pour cette raison et puisque Gradle permet de rapidement définir des tâches, une tâche sera ajoutée et portera le nom : « integration-tests ». Cette tâche s'exécutera après celle des tests unitaires et s'occupera exclusivement des tests d'intégrations.

2.2.1 Tests unitaires

Les tests unitaires sont particulièrement utiles pour assurer la qualité des composantes singulières du système. Ces tests sont habituellement rapides à exécuter et ne devraient pas dépendre de systèmes externes. Avec Java, le framework par excellence pour ce genre de tests est JUnit.

Ce framework offre une multitude d'outils pour tester ces classes et leurs fonctions. Il est extrêmement mature et bien documenter. De plus, il s'intègre très bien avec Spring et permettra de facilement tester plusieurs composantes du système d'arrière-plan.

Ce framework sera utilisé en coopération avec Mockito et Hamcrest. Le premier permet de « mocker », c'est-à-dire : remplacer un objet par une copie et définir ce que l'on attend de cette copie. Le « mocking » est un principe abondement utilisé dans les tests unitaires de système Java puisqu'il arrive souvent qu'une composante travaille à l'aide d'autres composantes. Dans un tel cas, les composantes requises seraient « mocker » afin de limiter l'étendue du test à la composante en cours de test.

Hamcrest quant à lui offre une série de méthodes pour effectuer la validation lors des tests. Cette validation (« assert ») est plus simple avec Hamcrest qu'avec JUnit et lorsqu'une validation échoue, le message de journal (log) est clair et précis quant à la raison de cet échec.

2.2.2 Tests d'intégration

Les tests d'intégration sont eux aussi importants dans le développement d'une application complexe. Ces tests valident les interactions entre les différents systèmes requis par le système testé (base de données, API, etc.).

Ces tests sont donc différents des tests unitaires. Ils requièrent que les systèmes externes requis soient fonctionnels pour leurs exécutions. Certains de ces tests peuvent même exiger le déploiement du système en cours de test. Par exemple, un service exposant un API REST pourrait définir une suite de tests qui exige que le système soit déployé. Cette suite de tests enverrait ensuite des requêtes sur les APIs exposés. Cette technique sera utilisée dans le cadre de ce projet.

À l'aide de Docker et de RestAssured, les tests d'intégrations valideront les APIs exposés par le système. L'application sera déployée en tant que conteneur Docker (voir Techniques de déploiement), et les tests utiliseront RestAssured pour soumettre de véritables requêtes HTTP sur l'application déployée.

Certains tests valideront également les interactions avec le système de gestion de données. Dans la même manière, le système de gestion de données (PostgreSQL) sera déployé dans un conteneur Docker, et une suite de tests validera les échanges entre le système et la base de données.

Vu leurs étendues, les tests d'intégrations sont nécessairement plus longs à exécuter que les tests unitaires. C'est pour cette raison qu'une tâche réservée exclusivement à l'exécution de ces tests a été ajoutée aux étapes de déploiement. Cette tâche pourra être ignorée durant le développement

CHAPITRE 3

SÉCURITÉ

3.1 Introduction

Le système d'arrière-plan est sécurisé afin de ne permettre qu'au propriétaire des ressources d'y avoir accès. Le propriétaire peut également donner l'accès à ses ressources à d'autres systèmes (client) qui pourrait le demander.

Ce processus de protection et de négociation des ressources utilise le protocole OAuth2. Ce protocole utilise la notion de jeton (« token ») afin d'identifier l'auteur d'une requête et de déterminer si celui-ci peut avoir accès à la ressource demandée.

Le protocole OAuth2 implique la coopération de plusieurs parties. Voici un tableau qui définit ces parties.

Tableau 1 - Définition des termes et parties reliés à OAuth2

Terme	**Définition**
Utilisateur (Resource Owner)	L'utilisateur est propriétaire de l'information protégée. L'utilisateur donne l'information permettant l'accès ou autorise un client à avoir accès à cette information.
Client	Le client est une application (web, mobile, etc.) qui permet l'affichage et la manipulation de l'information. Chaque client possède un identifiant et un secret. L'identifiant est connu, mais le secret devrait rester confidentiel. Il existe des clients qui ne peuvent conserver ces informations confidentielles, dans ce cas, ils n'utiliseront pas de secret.

	Les clients sont utilisés par les utilisateurs pour interagir avec le système.
Serveur d'autorisation	Le serveur d'autorisation est responsable d'identifier un requérant qu'il soit un client ou un utilisateur. Lorsque cette vérification est complétée, un jeton est transmis au requérant.
Serveur de ressources	Le serveur de ressources possède l'information de l'utilisateur. L'information est protégée et seul un jeton peut y donner accès. Le serveur de ressources est responsable d'effectuer la validation du jeton et des permissions associés à ce dernier.
Jeton (« token »)	Le jeton a une durée de vie limitée et il est associé à un seul utilisateur ou un client.

Afin d'avoir accès à l'information, l'utilisateur ou le client (qu'il utilise) doit récupérer un jeton. Ce jeton sera validé à chaque requête. Cette validation varie dépendamment des besoins en sécurité.

Un avantage important du protocole OAuth2 est qu'il est possible pour un utilisateur de donner l'accès à ses ressources à un système externe (client) sans que ce dernier ne connaisse ses informations de connexion. Il existe plusieurs types d'échanges qui permettent la récupération du jeton.

3.1.1 Échange « Authorization Code »

L'échange illustré ci-dessous s'appelle : « authorization_code ». Ce type d'échange devrait être utilisé lorsque le client est de confiance et que les informations de connexions

du client ne sont pas accessibles. Ainsi, le client peut interroger le serveur de ressource en tant qu'un utilisateur. L'échange se déroule habituellement ainsi :

1. L'utilisateur utilise un client (site web) pour accéder au service (système de covoiturage)
2. Le client doit demander la permission à l'utilisateur pour pouvoir récupérer ses informations auprès du système de covoiturage
    a. Initie le processus avec le serveur d'autorisation
    b. Demande à l'utilisateur sa permission
3. L'utilisateur donne la permission
    a. Confirme son identité auprès du serveur d'autorisation
    b. Confirme au serveur d'autorisation qu'il permet au client d'accéder à son information
    c. Le serveur d'autorisation génère un code d'autorisation
4. À travers une redirection, l'utilisateur transmet le code d'autorisation au client
5. Le client récupère le jeton d'accès
6. Le client échange son code d'autorisation avec un jeton d'accès
7. Le client communique avec le serveur de ressource pour récupérer l'information désirée

La figure ci-dessous montre cet échange. Ici, l'utilisateur accepte de donner la permission au client et aucune erreur ne survient au cours du processus.

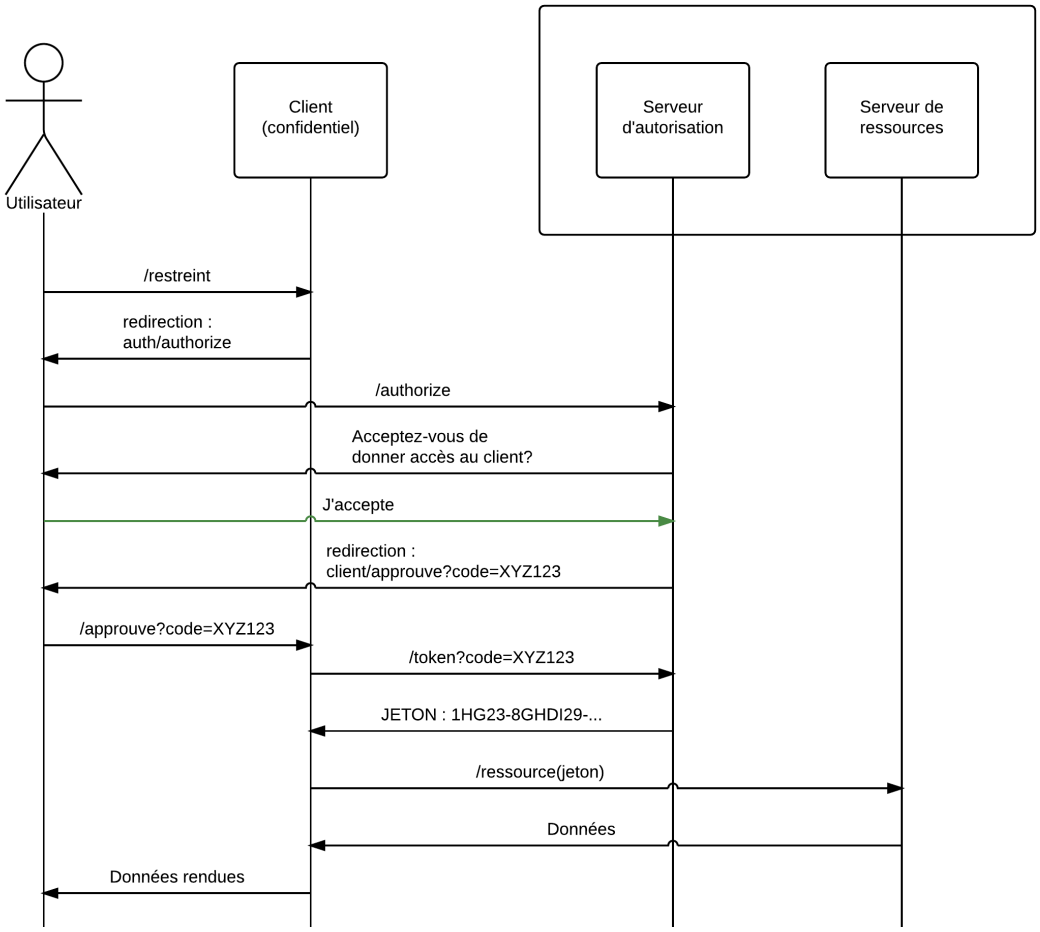

Figure 3-1 - Échange OAuth2 « Autorization Code »

3.1.2 Échange « Implicit »

L'échange « Implicit » est optimisé pour les applications web (JavaScript) qui s'exécutent dans le navigateur de l'utilisateur. Il est très similaire à l'échange « Authorization Code », cependant, le jeton d'accès est retourné directement dans la phase d'autorisation. Dans cet échange, le client n'est pas authentifié.

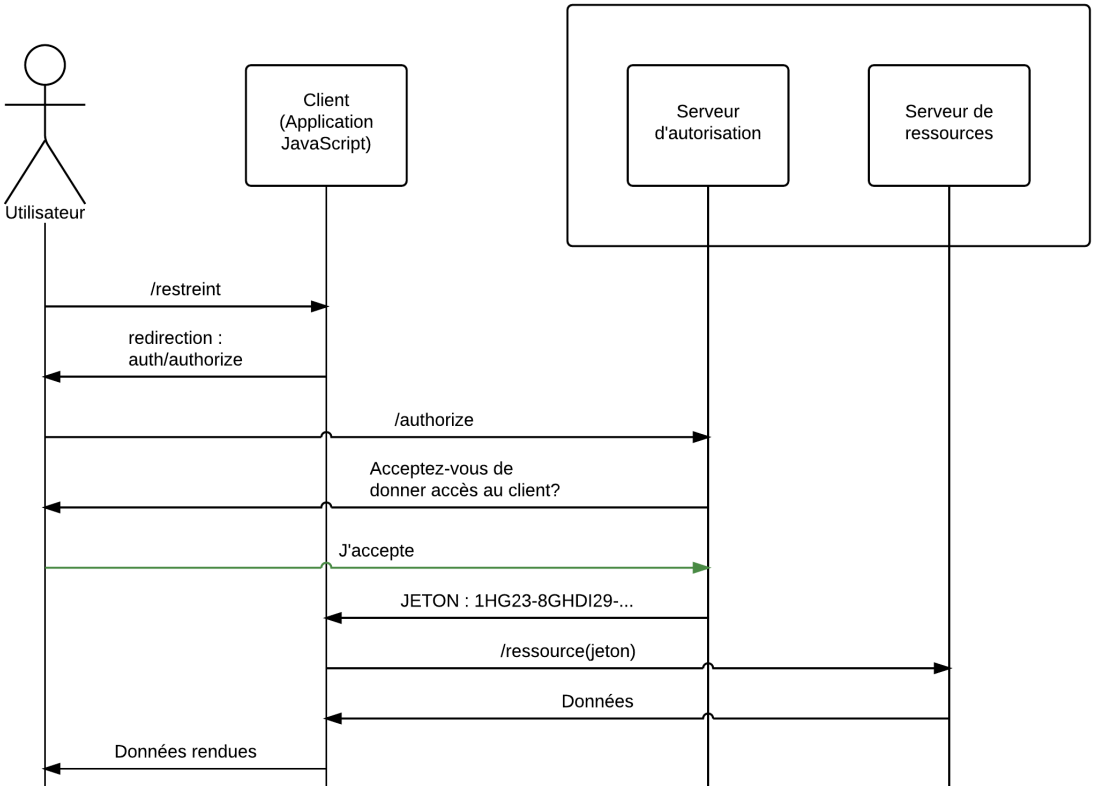

Figure 3-2 – Échange OAuth2 « Implicit »

Ce type d'échange permet une interaction plus fluide entre l'utilisateur et le client puisque moins d'aller-retour sont nécessaires. Par contre, puisque le jeton d'accès peut être exposé, il est nécessaire de prendre des précautions lorsque cet échange est utilisé. Dans plusieurs cas, le jeton d'accès aura une durée de vie très courte et devra être renouvelé régulièrement.

3.1.3 Échange « Resource Owner's Credentials »

L'échange « Resource Owner's Credentials » est simple. Dans cet échange, le client demande les informations de connexions de l'utilisateur. Ces informations sont utilisées afin de récupérer directement un jeton d'accès.

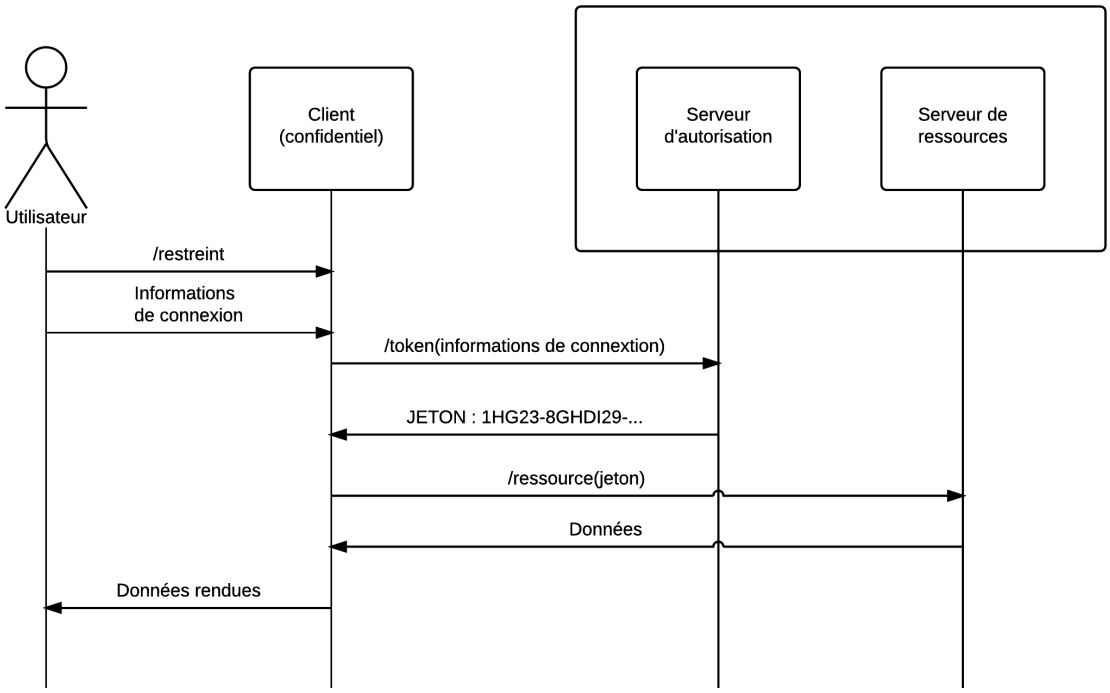

Figure 3-3 - Échange OAuth2 « Resource Owner Credentials »

Ce type d'échange ne devrait être utilisé que lorsqu'il existe un niveau de confiance élevé entre l'utilisateur et le client. Dès la réception du jeton, le client devrait se débarrasser des informations de connexion de l'utilisateur.

De plus, ce type d'échange peut être utilisé à la fois lorsque les informations de connexion du client sont confidentielles et à la fois lorsqu'elles ne le sont pas. Dans le premier cas, le serveur d'autorisation exigera l'authentification du client.

3.1.4 Échange « Client Credentials »

Il existe également un échange qui permet au client d'effectuer des opérations avec le serveur de ressource en son propre nom. Dans ce cas, le type d'échange à utiliser est le « Client Credentials ».

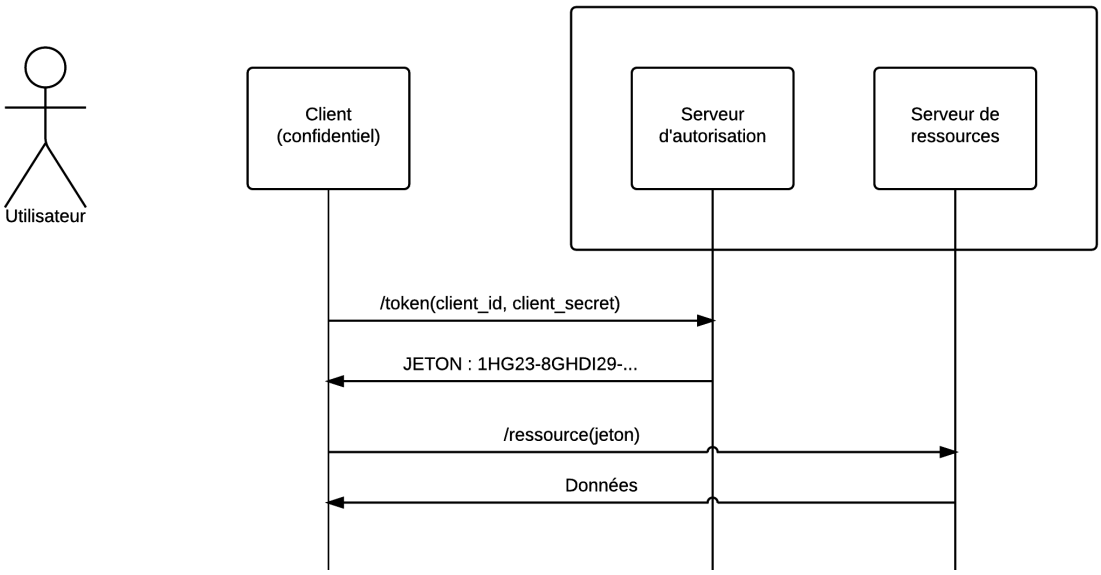

Figure 3-4 - Échange OAuth2 « Client Credentials »

Ce type d'échange exige que le client soit confidentiel. Lors de l'échange, le client échange ses informations de connexion (client_id et client_secret) pour un jeton d'accès.

3.1.5 Échange de rafraichissement

Dans le cas où le client a déjà en sa possession un jeton d'accès, mais que ce dernier est expiré, il est possible de le rafraichir sans procéder à un échange complet. Dans ce cas, le jeton de rafraichissement sera utilisé. Puisque ce ne sont pas tous les échanges qui fournissent le jeton de rafraichissement, il n'est pas toujours possible de rafraichir un jeton d'accès de cette façon.

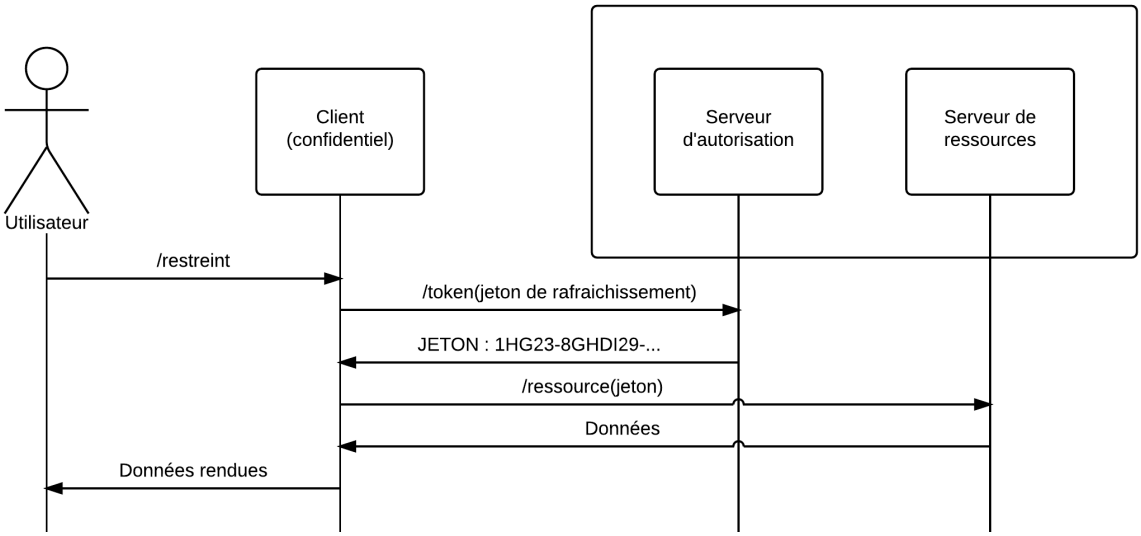

Figure 3-5 - Rafraichissement du jeton d'accès

3.2 Réponse

Habituellement, la réponse reçue lors de cet échange comporte quatre parties importantes et ressemble à ceci :

Tableau 2 - Réponse échange OAuth2

```
{
       "access_token":"2YotnFZFEjr1zCsicMWpAA",
       "token_type":"example",
       "expires_in":3600,
       "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
 }
```

La réponse contient habituellement quatre éléments importants :

- access_token : jeton d'accès utilisé lors de la communication avec le serveur de ressource
- token_type : type de jeton (Bearer dans le cadre du projet)
- expires_in : nombre de secondes avant l'expiration du jeton
- refresh_token : ce jeton est utilisé pour rafraichir le jeton d'accès lorsqu'il expire

Il est important de noter que cette réponse variera selon l'échange utilisé.

3.2.1 Jeton d'accès

Dans le cadre de ce projet, le jeton d'accès est un JSON Web Token (JWT). Ce type de jeton est un conteneur. Le jeton est encodé et apparait comme une série de caractère aléatoire. Cependant, à l'aide de la clé, il est possible de décoder le jeton et de récupérer l'information qu'il contient.

Ceci représente un avantage important au niveau de la performance. S'il existe des informations concernant le client ou l'utilisateur qui doivent être récupérés lors de la réception du jeton, le serveur d'autorisation n'a qu'à placer ces informations dans le conteneur avant de l'encoder.

À la réception, le serveur de ressource n'a qu'à décoder le jeton et il sera en mesure de récupérer l'information. Ceci évite que le serveur de ressource aille besoin d'accéder à une base de données, ou de tout autre méthode de stockage, à chaque réception de jeton. Dans le cadre de ce projet, le jeton contient le nom du tenancier. Cette information est intrinsèquement reliée à l'identificateur client utilisé lors de l'échange.

CHAPITRE 4

CONCEPTION DU SCHÉMA DE DONNÉES

4.1 Modèle conceptuel

À partir des différents cas d'utilisation, il n'est pas très complexe de modéliser le stockage d'information d'un système de covoiturage. Les cas d'utilisation sont définis dans le SRS (voir en annexe), mais les voici à titre de rappel :

- Inscription et authentification
- Publication et réservation d'un transport
- Recherche d'un transport
- Ajouté/modifié/supprimé d'une alerte
- Ajouté/modifier/supprimer d'une revue

Ces cinq cas d'utilisation représentent bien l'utilisation de base du système de covoiturage.

4.1.1 Entités

Les cas d'utilisations permettent d'identifier rapidement les entités qui devraient être stockées afin d'assurer le bon fonctionnement du système. De plus, il est trivial d'extraire les relations qui existent entre ces différentes entités.

4.1.1.1 Utilisateur

L'utilisateur est la représentation de l'usager dans le système. C'est une entité essentielle au bon fonctionnement du système puisque sans les utilisateurs, le système est inutile. Voici les attributs qui sont pertinents concernant l'utilisateur :

- nom et prénom
- âge
- numéro de téléphone

- courriel
- mot de passe
- date d'enregistrement dans le système

Dans le système, l'utilisateur peut occuper trois rôles : utilisateur, conducteur, et passager. Un conducteur peut publier des transports et posséder des véhicules. Le passager peut réserver des transports. De plus, le conducteur et le passager peuvent également rédiger des revues et/ou être le sujet de revues par d'autres utilisateurs.

4.1.1.2 Transport

Le transport correspond au déplacement à bord du véhicule d'un conducteur. Les transports sont publiés par des conducteurs. Les utilisateurs qui désirent se déplacer doivent trouver un transport qui correspond à leurs besoins. Un transport possède les attributs suivants :

- date de départ
- point de départ et d'arrivé (coordonnées GPS (latitude et longitude))
- prix

Le transport se fait à bord d'un véhicule. Il est nécessaire de conserver certaines informations concernant le véhicule afin d'aider les utilisateurs à identifier le conducteur une fois au point de départ.

4.1.1.3 Véhicule

Les informations conservées à propos du véhicule sont génériques et permettent une identification visuelle rapide. Les attributs d'un véhicule suivants sont conservés :

- marque et modèle
- année
- couleur
- plaque (numéro immatriculation)

Deux véhicules ayant des informations identiques peuvent être enregistrés au nom de deux conducteurs distincts. Ainsi, si un couple est propriétaire d'un seul véhicule, chaque individu de ce couple peut être un conducteur dans le système et offrir des transports.

4.1.1.4 Revue

Lorsqu'un passager réserve un transport, il peut rédiger une revue à propos du conducteur. Le conducteur est aussi en mesure de rédiger une revue pour chacun de ses passagers. Une revue possède les attributs suivants :

- date de la revue
- message (facultatif)
- une liste de facteurs d'évaluation

4.1.1.5 Facteur d'évaluation

Un facteur d'évaluation représente un aspect de la revue ainsi que son évaluation. Le facteur d'évaluation possède les attributs suivants :

- type
- valeur

La valeur est un pourcentage correspondant à la performance de l'utilisateur selon le type évalué. Un exemple de facteur d'évaluation pourrait être la ponctualité ou encore la conduite.

4.1.1.6 Alerte

Dans le système, les alertes sont utilisées par les passagers pour les informer qu'un transport similaire à ce qu'il recherche vient d'être affiché. Une alerte possède les attributs suivants :

- date et heure de départ
- point de départ et d'arrivée (coordonnées GPS (latitude et longitude))
- actif, SMS, courriel (vrai/faux)

Cependant, un système de notification sera également utilisé pour informer les conducteurs lorsque des passagers réservent leurs places dans un transport. Il en va de même pour les annulations.

4.1.2 Diagramme

Afin d'indiquer les différentes relations entre les entités, un diagramme est approprié. Le diagramme suivant montre donc le schéma conceptuel du système de covoiturage :

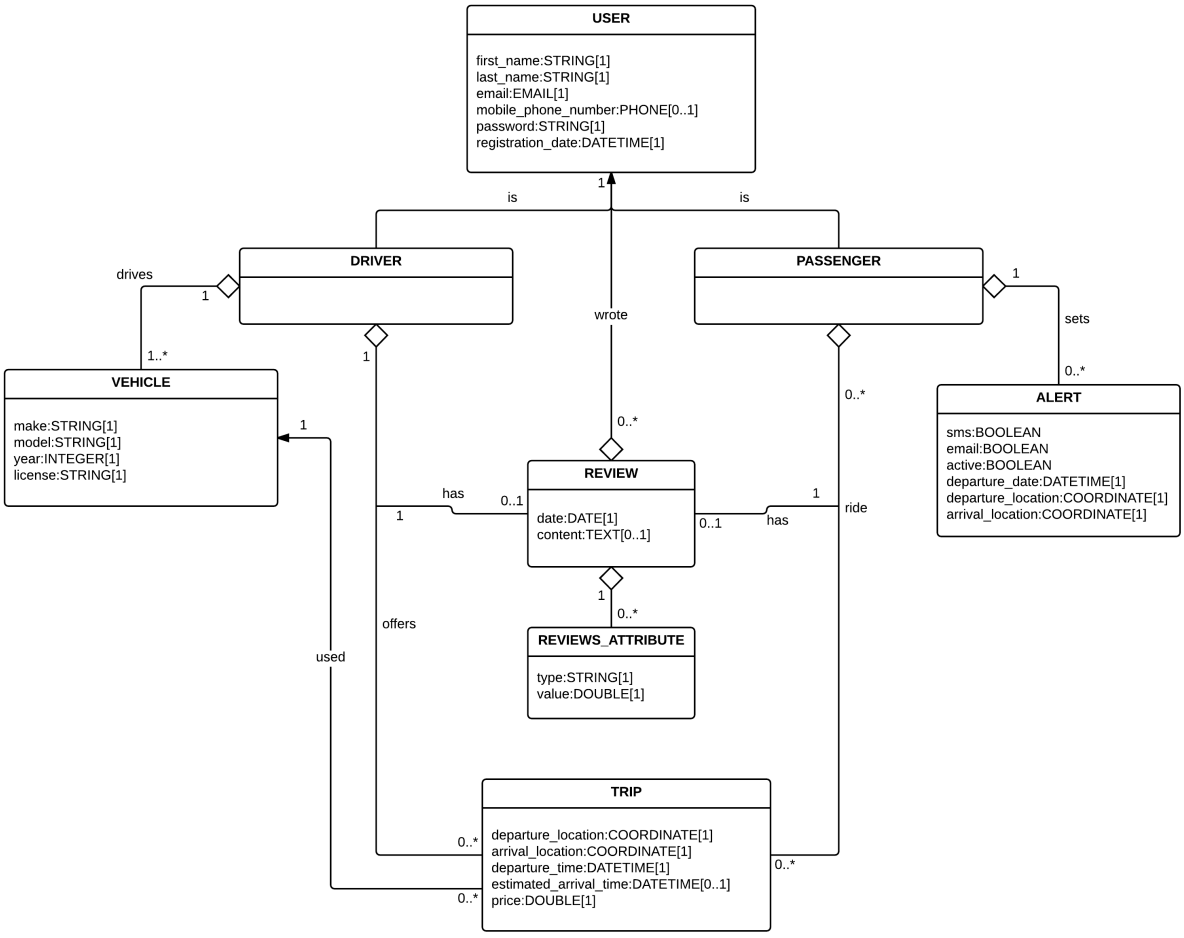

Figure 4-1 - Modèle conceptuel

4.2 Modèle relationnel

À partir du modèle conceptuel, il est peu complexe de produire un modèle relationnel. La différence entre le modèle relationnel et le modèle conceptuel est simple : la version relationnelle du modèle de données est directement rattachée à la technologie qui sera utilisée.

Dans le cadre de ce projet, il a été déterminé que PostgreSQL serait le SGBD de choix (*voir* Technologies pour le système de gestion de base de données).

Le diagramme suivant montre donc une version du modèle conceptuel de données normalisée et adaptée au SGBD PostgreSQL:

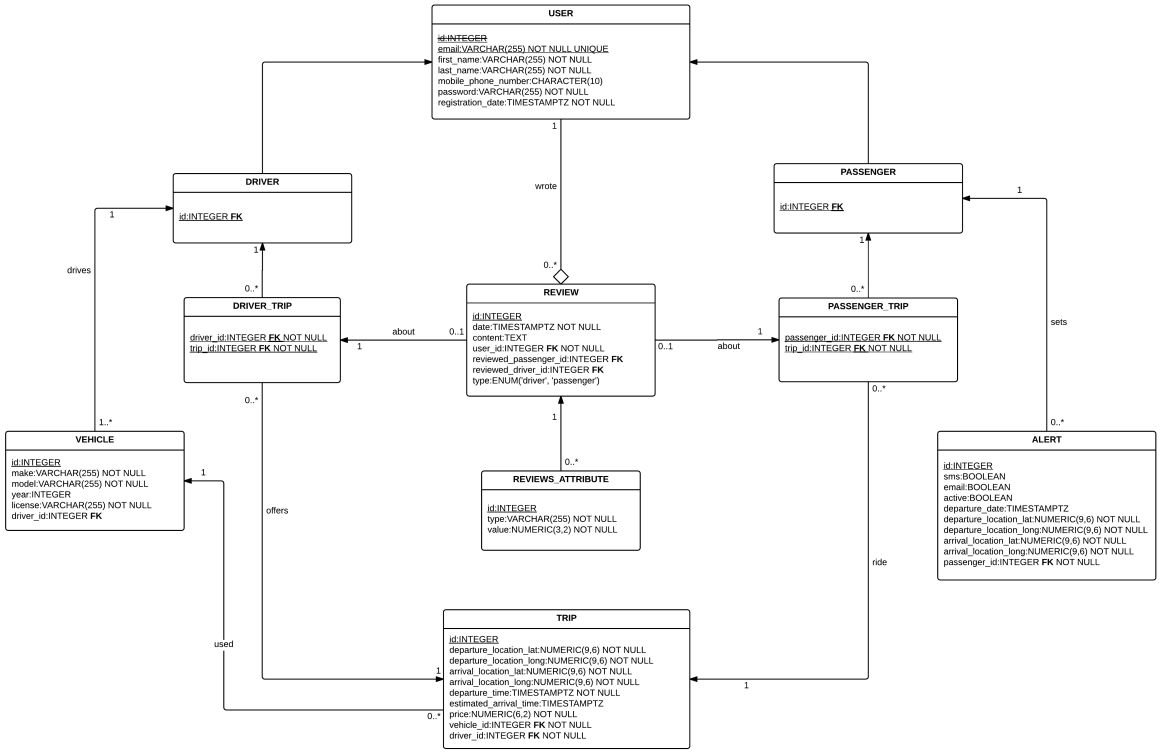

Figure 4-2 - Modèle relationnel

CHAPITRE 5

ARCHITECTURE D'INFRASTRUCTURE

5.1 Introduction

Un système d'arrière-plan est un système complexe. Afin de tirer un maximum de profit du système et de l'équipe responsable de son développement, plusieurs décisions concernant l'architecture de l'infrastructure devront être prises.

Ces décisions affecteront la qualité du système livré, la rapidité avec laquelle il peut être livré, ainsi que la stabilité du processus de livraison. Les prochaines sections expliquent certaines décisions prises afin de maximiser ce processus.

5.2 Déploiement automatisé

5.2.1 Définition

Le déploiement automatisé est une approche du génie logiciel qui consiste à automatiser la livraison d'un système de façon complètement automatisé. Cette approche est utilisée afin d'augmenter l'efficacité de l'équipe de développement d'un système tout en réduisant le taux d'erreur et en augmentant la qualité des livraisons.

La livraison d'un système comprend plusieurs tâches. Ces tâches doivent être exécutées dans l'ordre et les artéfacts obtenus à la sortie d'une tâche servent habituellement de données d'entrées de la tâche subséquente. Parmi ces tâches, on retrouve :

- compilation
- tests unitaires
- tests d'intégration
- déploiement

Bien sûr, cette liste n'est pas exhaustive et elle varie d'un milieu à l'autre. L'exécution automatisée de ces tâches affectera différents aspects de la construction des systèmes.

5.2.2 Tâches

Avant d'entamer l'automatisation du processus de déploiement, il est important de bien comprendre le rôle de chacune des tâches de ce processus. Comme il est indiqué plus haut, les tâches doivent s'exécuter dans un ordre prédéterminer. Cet ordre est nécessaire étant donné la nature des tâches.

Les tâches seront définies dans la prochaine sous-section selon l'ordre d'exécution. Par la suite, les implications qui découlent de ces tâches, ainsi que leurs impacts seront expliqués.

5.2.2.1 Compilation

Dans le monde du logiciel, le principal livrable est un système logiciel. À la base de tout système logiciel se trouve le code source. Ce code source est la porte d'entrée du développeur pour définir le comportement du système qu'il développe.

Cependant, le langage utilisé est une abstraction permettant de définir ce comportement. Ce langage est habituellement converti dans une autre forme qui, elle, peut être exécutée par le système hôte. Cette transformation d'un langage vers une autre forme, souvent exécutable, s'appelle : la compilation.

Il est important de noter de la compilation n'est pas nécessaire pour l'exécution de tous les langages. Certains langages sont interprétés. Cela signifie qu'ils sont interprétés par un autre programme, lors de l'exécution, afin de déterminer les opérations qui seront exécutées. Cependant, cette interprétation à un coût qui peut être non-négligeable.

Dans le cadre du projet, Java est le langage utilisé et le code source produit dans ce langage doit être compilé. Cependant, la compilation d'un programme Java ne donne pas directement un exécutable, mais plutôt du Java Byte Code qui sera exécuté par la Java Virtual Machine (JVM) qui se trouve sur le système hôte.

Pour compiler du code source Java, le programme *javac* est requis. Cependant, il est habituel d'utiliser des outils de plus haut niveau pour compiler un programme Java. Des outils comme Maven et Gradle sont particulièrement populaires.

Dans le cadre de notre projet, Gradle est utilisé et la tâche qui permet d'effectuer la compilation du système est « compileJava ». Cependant, cette tâche est assez limitée puisqu'elle ne fait que compiler les fichiers avec l'extension .java et ignore complètement les différentes ressources qui pourraient être requise par le système (fichiers de configurations, etc.).

C'est pour cette raison qu'habituellement, les outils de gestion de projets comme Maven et Gradle offrent des tâches globales qui rassemblent toutes ces petites sous-tâches qui devraient être exécutées. La tâche Gradle qui permet ceci est « classes ».

Une fois compilé, un projet java peut être exécuté. Cependant, le processus de compilation produit une grande quantité d'artéfacts et il n'est pas pratique de manipuler directement ceux-ci. Java propose donc un contenant pour livrer un système Java : le jar. Ce type de fichier (.jar) n'est qu'un contenant dans lequel on retrouve l'ensemble des artéfacts de compilation. Les *Jars* peuvent être des systèmes exécutables ou des librairies utilisés par d'autres systèmes Java. Gradle permet encore une fois de réaliser cette tâche. À l'aide de la tâche « jar », un fichier contenant l'ensemble de programmes est produit.

Une fois la compilation terminée, il est important de tester les artéfacts qui ont été générés.

5.2.2.2 Tests unitaires

La tâche suivant la compilation est l'exécution des tests unitaires. Les tests unitaires sont une partie du code source d'un système qui ne sert qu'à tester le comportement des composants du système. Évidemment, cette partie du code source doit elle aussi être compilée.

Donc après la compilation du système, les tests sont compilés et exécutés. Évidemment, ces deux autres tâches sont aussi gérées par Gradle avec « testClasses » et « test ».

L'exécution des tests produits des artéfacts qui varient selon le framework utilisé. Dans le cadre du projet, JUnit est utilisé et l'exécution de tests JUnit produit plusieurs artéfacts. Parmi ces artéfacts, on retrouve des rapports d'exécution et de couvertures, les journaux (logs) de chaque test, etc. De plus, la tâche d'exécution des tests peut être un succès ou un échec et ce résultat global peut être utilisé afin de contrôler le comportement du processus en entier. Par exemple, il est normal de voir un processus de déploiement automatisé s'interrompre lorsque le résultat d'exécution des tests unitaires est un échec.

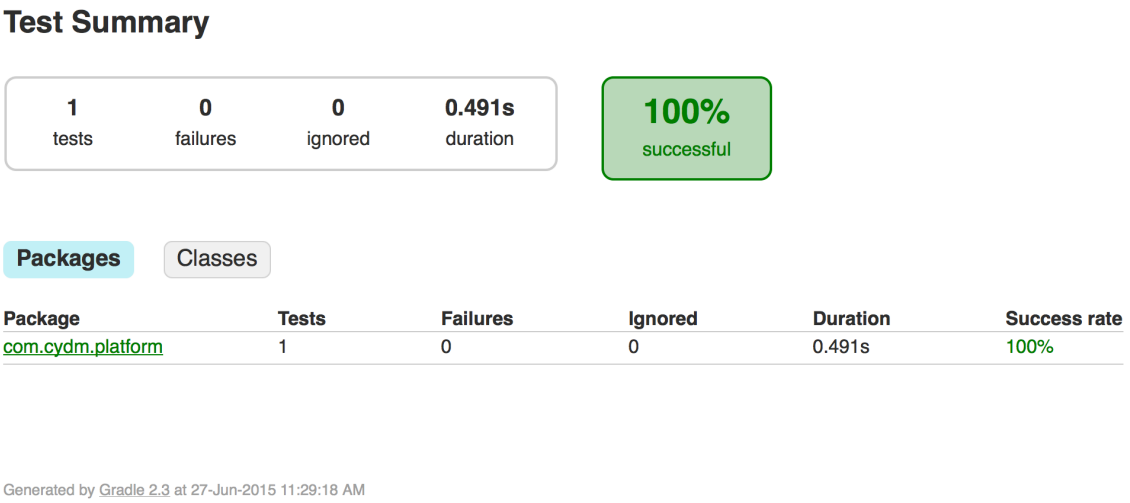

Figure 5-1 – Capture d'écran d'un rapport d'exécution de tests

Dans le cadre du projet, le succès de l'exécution des tests est requis afin de passer à la tâche subséquente : l'exécution des tests d'intégrations.

5.2.2.3 Tests d'intégrations

Une fois l'exécution de tests unitaire terminée avec succès, l'exécution de tests d'intégration peut être entamée. Les tests d'intégration sont fondamentalement différents des tests unitaires et c'est pourquoi il existe deux tâches distinctes pour l'exécution de ces types de tests.

Les tests d'intégration visent à valider le comportement d'un système lorsqu'il interagit avec d'autres composants du système. Un exemple simple et commun est celui des tests qui interroge une base de données. Dans un contexte de tests unitaires, les interactions avec la base de données sont remplacées afin d'isoler la portée des tests. Dans un contexte de tests d'intégration, le test consiste à valider l'échange entre la base de données et le système. Dans ce test, une connexion serait établie avec une véritable base de données.

Il est donc important de noter que l'exécution de ces tests est beaucoup plus laborieuse que celle des tests unitaires. L'exécution d'une telle suite de tests nécessite que l'ensemble des composants requis soit en fonction.

Au même titre que l'exécution des tests unitaires, l'exécution des tests d'intégration génère plusieurs artéfacts qui permettent de visualiser rapidement les caractéristiques de la suite de test. Lorsque cette tâche s'exécute avec succès, la dernière tâche peut-être entreprise.

5.2.2.4 Déploiement

La dernière tâche du cycle est le déploiement. Cette tâche est habituellement exécutée seulement lorsque toutes les tâches précédentes ont été exécutées avec succès. À cette étape, la nouvelle version du système qui sera déployée satisfait les différentes exigences et n'affecte pas négativement les fonctionnalités déjà existantes (régression).

Le déploiement peut-être complètement automatisé. Cela signifie que la tâche est entreprise automatiquement. Dans une telle situation, le système d'intégration continu (*voir Intégration continue*) se chargera de déployer la nouvelle version du système dans l'environnement désigné. Si ce déploiement est totalement automatisé, une série de mesure devra être mise en place afin d'alerter les administrateurs du système si un problème survient. Il est possible aussi que des opérations de retour en arrière ou de déploiement partiel soient employées dans ce genre de tâches.

Cependant, il arrive très souvent que cette étape du processus soit supervisée par un intervenant humain. Dans un tel cas, lorsque la tâche précédente est accomplie avec succès. Une action manuelle est nécessaire pour déployer la nouvelle version du système. Cette action peut-être le lancement d'un script qui effectue le déploiement, une série d'opérations prédéfinies où le système est manuellement déployé dans l'environnement ciblé, etc. Dans ce cas, l'action est supervisée et si un problème survient, la personne impliquée devra gérer le problème.

5.2.3 Implications

Les tâches mentionnées plus haut définissent les étapes du processus de déploiement. Pour exécuter ce processus, différents outils peuvent être utilisés. Dans un projet où l'équipe de développement est très petite, l'exécution peut être manuelle. Cependant, lorsque l'équipe grandit et que le nombre de livraisons augmente, il devient difficile et inefficace de procéder de façon manuelle.

Dans le cadre de ce projet, voici comment se déroule le processus de déploiement automatisé :

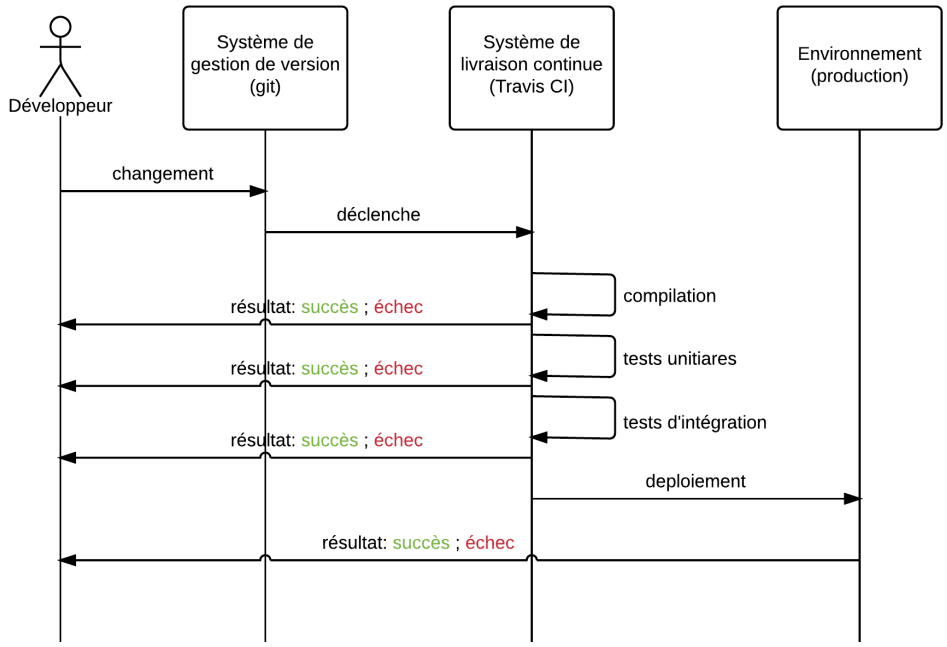

Figure 5-2 - Déploiement automatisé

Dans la figure ci-dessus, l'importance de deux outils est mise en évidence : le système de gestion de version et le système d'intégration continue. La séquence des évènements est décrite en détail dans les sections suivantes.

5.2.3.1 Système de gestion de version

Le déclenchement d'une séquence de déploiement commence par un changement au niveau du code source du système. Ce changement est fait d'abord fait localement par le développeur. Lorsque le développeur est satisfait, il peut confirmer son changement dans le SGV. Dans le cadre de ce projet, Git est utilisé. Dans Git, cette action ce traduit habituellement par un « commit » qui est poussé (« push ») dans le « repository » du système.

Il est important de noter qu'il existe bien d'autre façon de faire. Dans certains milieux, une revue devra être effectuée (recommandé) avant que les changements du développeur soient intégrés dans le code source du système.

Suite à ce changement, le système d'intégration continu (SIC) est informé. Encore une fois, plusieurs techniques existent : le SGV exécute une action suite à un changement au code, le système d'intégration continue interroge le SGV périodiquement afin de détecter un changement, etc.

Dans tous les cas, le SIC prend la relève.

5.2.3.2 Système d'intégration continue

À cette étape, le système d'intégration continue (SIC) est informé d'un changement dans le code source. Il peut donc démarrer l'exécution du processus de déploiement automatisé.

L'utilisation ici d'un outil comme Gradle ou Maven est très avantageuse. Ces outils offrent des tâches qui exécutent le processus en entier. Dans le cadre de ce projet, Gradle a été sélectionné et la tâche à exécuter est « build ». Cette tâche compilera et exécutera les tests unitaires et d'intégration. Évidemment, si l'une de ces exécutions n'est pas un succès, le processus sera interrompu et le développeur sera informé de l'échec. Si toutes les étapes se déroulent bien, le déploiement sera effectué.

5.3 Système multitenanciers

5.3.1 Définition

Dans un système multitenanciers (SMT), une même instance se comporte différemment selon le tenancier qui l'utilise. Lors de l'exécution d'une requête, le comportement du système et la source des données sont déterminés selon la source de la requête. L'avantage d'un SMT est qu'il existe un seul système pour plusieurs tenanciers. Il existe donc un partage de ressources important.

Un *tenancier* représente un ensemble d'utilisateurs ayant dans similitude. Par exemple, dans un système de covoiturage, les tenanciers peuvent être diverses compagnies qui

désirent offrir ce service à leurs employés. Dans ce cas, pour chaque tenancier, il existe habituellement une configuration, une source de données, un ensemble d'utilisateurs, des fonctionnalités particulières, etc.

Un SMT est intrinsèquement plus complexe qu'un système à client unique. Cette complexité se traduit à travers différentes facettes du système :

- conception système de gestion de base de données
- sécurité
- configuration
- mise à jour du schéma de bases de données

Ces différences de complexité seront expliquées dans la prochaine section.

5.3.2 Implications

La conception d'un SMT présente différents défis conceptuels qui ne sont habituellement pas présents dans un système web standard.

5.3.2.1 Gestion de la persistance

Dans un SMT, il existe trois façons différentes d'assurer la persistance des données :

- une base de données dédiée par tenancier
- un schéma par tenancier
- une base de données partagée entre tenanciers avec discrimination

Dans le premier cas, chaque tenancier possède sa propre base de données. Cela permet l'isolation complète des données d'un tenancier à un autre. Cependant, cela exige que l'application soit en mesure d'utiliser la bonne source de données selon le tenancier qui utilise le système. Dans la plupart des cas, il sera nécessaire d'avoir un « pool » de connexion par tenancier. De plus, la gestion du schéma de ces différentes bases de données est plus complexe (*voir* Mise à jour du schéma de base de données).

Dans le second cas, une seule instance de base de données est utilisée. Habituellement, en Java, la connexion vers la base de données spécifie le schéma qui sera utilisé. Cependant, il est possible, dans certains cas, d'établir la connexion et d'ensuite préciser le schéma à utiliser. C'est cette technique qui est utilisée avec une approche multischémas.

Dans le dernier cas, une seule base de données et un schéma sont requis. L'application n'a donc qu'une seule source de données à gérer. Cependant, afin de récupérer l'information d'un seul client à la fois, une colonne discriminante est placée dans les tables où cela est nécessaire. Cette colonne contient l'identifient d'un tenancier. Chaque identifiant représente un seul tenancier.

Dans le cadre de ce projet, la deuxième technique sera utilisée. Il existera donc un schéma par tenancier dans une même base de données. Les ressources requises pour opérer le système seront ainsi réduites.

5.3.2.2 Sécurité

La sécurité des SMT doit être prise au sérieux. À l'instar d'un système traditionnel à tenancier unique, les SMT regroupent l'information de plusieurs tenanciers à un seul endroit.

Dans le cas d'une brèche, un attaquant aura accès aux données de plus d'un client. Ce type de système devrait par conséquent être conçu avec un accent important sur la sécurité des données. Les différentes permissions des bases de données devraient donc être gérées avec précaution.

Les différentes décisions concernant la sécurité qui ont été prises dans le cadre de ce projet se retrouvent dans la section **Error! Reference source not found.**.

5.3.2.3 Configuration

Puisque le même système est utilisé par plusieurs tenanciers et que ces différents clients n'ont pas les mêmes exigences fonctionnelles et non fonctionnelles, le système doit être hautement configurable.

Dans ce système, les composantes qui ont un comportement variable d'un client à l'autre devront connaitre la configuration du tenancier qui pose une action sur le système. Typiquement, les configurations des tenanciers seront accessibles dans la base de données. De cette façon, il est trivial de modifier la configuration et de changer le comportement de l'application en cours d'exécution.

5.3.2.4 Mise à jour du schéma de base de données

Dans un SMT, où chaque tenancier possède son propre schéma, il peut devenir lourd d'appliquer les migrations de schémas pour tous les clients. De plus, dans une architecture où plusieurs systèmes interagissent avec les mêmes sources de données, il est complexe et très arbitraire d'attribuer la responsabilité de gérer le schéma à un système plutôt qu'un autre.

Dans ce cas, il devient intéressant d'extraire la fonctionnalité de l'outil de migration dans une composante à part entière. Cette composante, le système de gestion de schéma, SGS, assure la migration d'un seul schéma sur plusieurs bases de données. Le SGS possède son propre SGV et son utilisation est indépendante des systèmes qui utilisent les bases de données sur lequel la composante opère, *voir* Système de gestion de schéma.

CHAPITRE 6

SYSTÈME DE GESTION DE SCHÉMA

6.1 Définition

La grande partie des systèmes informatiques Web utilise une base de données afin de préserver l'information générée. Il existe une grande variété de types de base de données sur le marché (voir Technologies pour le système de pour un aperçu).

Dans bien des cas, ces bases de données structurent l'information selon un schéma prédéfini (*voir* Conception du schéma de données). Les bases de données utilisant un schéma sont souvent de type relationnel. Ces bases de données sont particulièrement adaptées pour des systèmes exigeant un grand nombre transactions. Ces bases de données peuvent aussi garantir l'intégrité des données en effectuant les transactions en respectant un ensemble de propriétés : Atomique, Consistent, Isolation, Durable (ACID). Ce type de base de données est donc très populaire et répandu.

Cependant, la maintenance du schéma d'une telle base de données peut être difficile. Lorsque des changements sont effectués sur le schéma, la base de données peut être inaccessible durant plusieurs instants. De plus, il peut être difficile de gérer l'historique du schéma d'une base de données.

Lorsqu'un système n'utilise qu'une seule source de données, il n'est pas très complexe de maintenir le schéma de celle-ci. Il est possible de procéder plusieurs façons distinctes :

- manuelle
- outil de migrations de schéma

La première approche est la plus simple, mais elle devient rapidement peu efficace. De plus, si une erreur survient, il est difficile de détecter la source de cette erreur.

La deuxième approche apporte plusieurs avantages. D'abord, un historique des migrations appliquées est conservé et souvent, les migrations sont immuables, c'est-à-dire qu'elles ne peuvent changer une fois qu'elles ont été appliquées. Ensuite, les interactions manuelles avec le schéma sont réduites au minimum ce qui réduit le potentiel d'erreur. Finalement, les migrations peuvent servir à la génération de base de données de test à la volée durant l'exécution des tests.

Le système de gestion de schéma (SGS) est un outil spécialisé qui permet la mise à jour et la maintenance du schéma d'une base de données.

6.2 Utilité

Habituellement, le fonctionnement d'un gestionnaire de schéma est simple. La première étape consiste à définir la ou les migrations qui devront être appliquées sur la base de données. Ensuite, l'exécution de l'outil se charge du reste.

Il est commun de retrouver ce genre de processus à l'intérieur même du système qui utilise la base de données. Dans un tel cas, l'exécution de la SGS est souvent faite au lancement de l'application. Chaque migration est appliquée sur le schéma de la base de données avec laquelle l'application interagit. Cependant, comme indiqué précédemment, il devient difficile de procéder ainsi lorsqu'un même système utilise plus d'une base de données.

Dans ce cas, extraire la fonctionnalité dans un système à part entière est une solution pratique. Ce système peut alors gérer un schéma pour plus d'une base de données. Cette extraction retire également la responsabilité de la gestion du schéma des différents systèmes qui interagissent avec les bases de données en question.

6.3 Fonctionnement

6.3.1 Survol

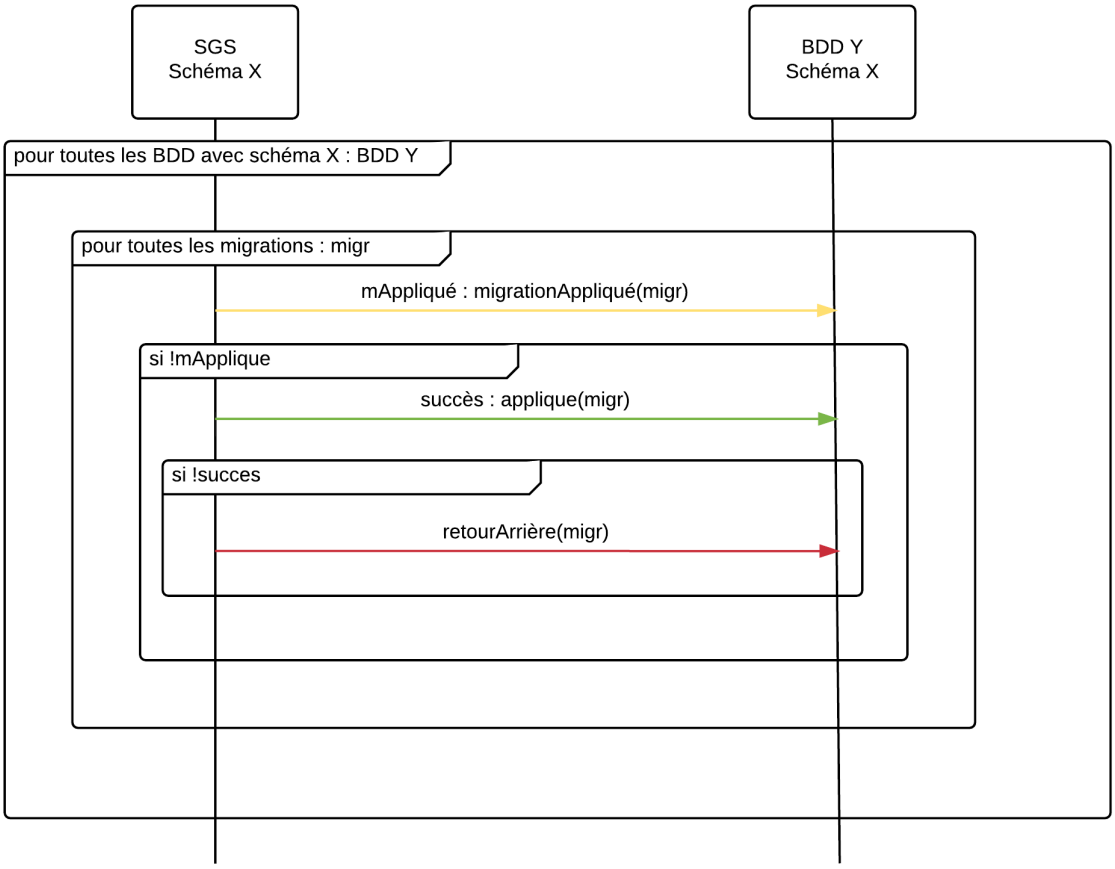

Figure 6-1 - Séquence d'exécution d'un SGS

Le diagramme semble complexe, mais, en réalité, le processus d'exécution d'un SGS est très simple. Pour chaque base de données utilisant le schéma géré, le système valide que chacune des migrations soit appliquée. Si une erreur survient à l'application d'une migration, un retour en arrière est effectué.

Dans le cadre de ce projet, cet outil est distribué dans un conteneur. La technologie utilisée ici est Docker. Plusieurs avantages sont directement liés à la « conteneurisation » du SGS :

- configuration simplifiée grâce aux liaisons de conteneurs

- historique
- génération de conteneur de base de données de tests

6.3.2 Configuration simplifiée

Docker permet de lier différents conteneurs entres-eux. Cette liaison expose les variables d'environnement du conteneur lié à l'intérieur du conteneur qui est lié. Par exemple, soit deux conteneurs : A et B. Le conteneur A défini la variable « MSG ». En démarrant le conteneur B avec une liaison à A, Docker définira une nouvelle variable d'environnement dans le conteneur B : « A_MSG » qui contient le contenu original.

À l'aide de cette liaison, le système opérant dans le conteneur peut utiliser les variables d'environnement en tant que valeurs de configuration.

6.3.3 Historique

Les images à la base des conteneurs Docker ont une propriété intéressante : l'immutabilité. Cette propriété garantit qu'un conteneur lancé à partir d'une même image sera toujours le même. Une fois qu'un changement est sauvegardé dans une image, une nouvelle image est créée.

À partir de ce principe, il devient simple de générer une nouvelle image à chaque modification dans le gestionnaire de schémas. Ainsi, il est toujours possible de régénérer le schéma comme il a été généré au départ.

6.3.4 Gestion de multiple base de données

En externalisant le processus de mise à jour de schéma, le schéma n'est plus directement relié à une base de données. Il est donc possible d'utiliser l'outil afin d'appliquer les migrations sur plusieurs bases de données ayant le même schéma ou sur une base de données ayant plusieurs fois le même schéma.

Le diagramme suivant montre deux SGS distincts en exécution. Chaque SGS gère un seul schéma et ce schéma peut être maintenu dans différentes bases de données (pour plusieurs tenanciers).

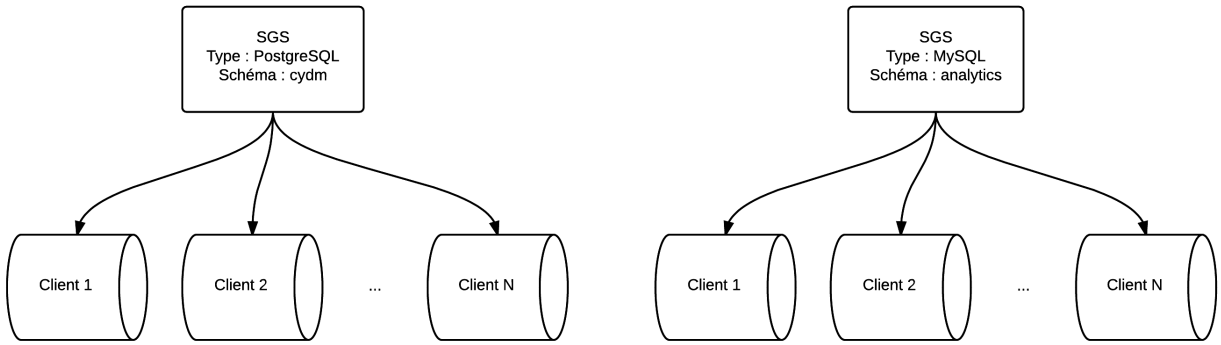

Figure 6-2 - Deux SGS gèrent différents schémas dans différentes BDD

6.3.5 Conteneurs de test

Un autre avantage intéressant des conteneurs dans le cadre du SGS est la génération de conteneur de test. À l'aide du SIC, il est trivial de configuré une tâche qui génèrera un conteneur contenant une base de données avec le schéma à la version spécifié qui pourra être utilisé dans le cadre de tests. Il est également possible de configurer ce conteneur afin qu'il contienne des données de tests.

Ce même conteneur pourra éventuellement servir à effectuer la validation des migrations avant qu'elles ne soient appliquées. Ainsi, à la création de nouvelles migrations, le SIC sera en mesure d'indiquer s'il existe des problèmes avec les migrations et prévenir l'exécution d'une migration erronée en production.

6.4 Limitations

Ce type de système apporte bien des avantages, cependant, il existe des limitations importantes qui doivent être considérées. Ces limitations affectent dans la plupart des cas la vélocité de développement ainsi que la complexité entourant une migration de schéma.

6.4.1 Retour en arrière

Dans le cadre de ce projet, la technologie utilisée afin de gérer les différentes migrations est Flyway. Cette technologie n'offre pas la possibilité d'effectuer un retour en arrière lorsqu'un problème survient à l'application d'une migration.

Cette limitation est justifiée par le simple fait qu'habituellement, lorsqu'un problème survient à l'application d'une migration, la base de données est dans un état instable. Dans ce cas, les probabilités que l'application d'une migration de retour en arrière soit un succès sont faibles.

De plus, une migration de retour en arrière assume que la migration originale a échoué en entier. Par contre, il est fort probable que la migration originale ait été composée de plusieurs instructions et que certaines d'entres-elles aient été appliquées avec succès.

Les migrations de schéma devront donc être bien testées avant d'être appliquées en production. De plus, il est impératif d'avoir en place un système de restauration éprouvé qui pourra être utilisé en cas de problème majeur.

6.4.2 SGBD supportés

L'utilisation de Flyway (indiqué plus haut) apporte une autre limitation. Cette technologie supporte un nombre limité de systèmes de gestion de base de données. Cette liste contient exclusivement des bases de données de type relationnel : Oracle, MySQL, SQL Server, etc.

Il n'est donc pas possible pour l'instant d'utiliser des SGBD qui ne sont pas supportés par Flway. Par exemple, bien que Cassandra soit une base de données NoSQL, il existe une notion de famille de colonne qui s'apparente à un schéma et qu'il serait intéressant de pouvoir gérer à travers le SGS.

6.4.3 Compatibilité

Puisque la gestion du schéma est externalisée dans un outil indépendant des applications qui utilisent le schéma, il est nécessaire que les migrations n'affectent pas les systèmes qui utilisent actuellement le schéma.

Par exemple, il est nécessaire de renommer une colonne dans une table. Dans ce cas, il existe sans aucun doute des systèmes qui seront impactés par le changement de colonne. Il est donc nécessaire d'effectuer les changements de façon graduelle afin de n'avoir aucun impact. Une première migration pourrait créer la nouvelle colonne et y copier les anciennes données. Ensuite, les systèmes affectés pourraient être mises à jour afin d'utiliser la nouvelle colonne. Et finalement, une deuxième migration pourrait synchroniser les colonnes et ensuite supprimer l'ancienne.

Ainsi, c'est la responsabilité du développeur qui s'occupe de la migration de ne pas affecter les systèmes en cours d'exécution. Bien sûr, à l'aide du SIC, il est possible pour le développeur de tester sa migration sur ces systèmes de façon automatisée,

CONCLUSION

Les besoins de déplacement de longue distance des gens ne sont pas près de diminuer. Il serait donc intéressant d'offrir une véritable solution au problème. Les consommateurs devraient avoir accès à un système de covoiturage efficace, simple et gratuit.

Dans le cadre de ce projet, un système d'arrière-plan a été développé. Ce système est l'une des composantes pour bâtir une telle solution. Ce rapport vise à documenter le processus de conception de ce système.

Le rapport présente les différents choix technologiques, les décisions concernant le déploiement, l'automatisation et la mise à jour des différents schémas de données. De plus, une brève introduction à OAuth2, protocole de sécurité utilisé dans le cadre du projet, y est présentée.

Ce système n'est pas très utile s'il n'est pas utilisé en coopération avec des applications clients. Ces différentes applications sont des systèmes en soi qui pourraient également entreprises dans le cadre d'un projet de fin d'études. Il existe actuellement un besoin pour plusieurs types d'application clients différents : Android, iOS, Windows Phone et application Web. La conception de chacun de ses clients apporterait une valeur ajoutée au système.

Il existe aussi des tâches à faire dans le cadre de ce projet : implémenté un réel moteur de recherche avec ElasticSearch, permettre aux conducteurs de définir des points intermédiaires dans leur déplacement. Malheureusement, le temps étant une ressource finie, ces tâches n'ont pu être entamées dans le cadre de ce cours.

LISTE DE RÉFÉRENCES BILIOGRAPHIQUES

Meteor, « Open-source platform for building top-quality web apps in a fraction of the time. ». En ligne. <https://www.meteor.com/>. Consulté le 2015-06-12

Node.Js, « Node.js® is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.». En ligne. <https://nodejs.org/>. Consulté le 2015-06-12

Grunt, « Grunt: The JavaScript Task Runner ». En ligne. <http://gruntjs.com/>. Consulté le 2015-06-12

NPM, « npm is the package manager for javascript ». En ligne. <https://www.npmjs.com/>. Consulté le 2015-06-12

Bower, « Bower manages all these frontend dependencies for you. ». En ligne. <http://bower.io/>. Consulté le 2015-06-12

Yeoman, « The web's scaffolding tool for modern webapps ». En ligne. <http://yeoman.io/>. Consulté le 2015-06-12

Wikipedia, « Java (programming language) ». En ligne. <https://en.wikipedia.org/wiki/Java_%28programming_language%29>. Consulté le 2015-06-12

Spring, « Spring helps development teams everywhere build simple, portable, fast and flexible JVM-based systems and applications. ». En ligne. <https://spring.io/>. Consulté le 2015-06-12

Hibernate, « Hibernate an open source Java persistence framework project. ». En ligne. <http://hibernate.org/>. Consulté le 2015-06-12

Spring Boot, « Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". ». En ligne. <http://projects.spring.io/spring-boot/>. Consulté le 2015-06-17

Gradle, «The Endgame Enterprise Build Automation Solution ». En ligne. <http://gradle.org/>. Consulté le 2015-06-17

JUnit, « JUnit is a simple framework to write repeatable tests. ». En ligne. <http://junit.org/>. Consulté le 2015-06-17

Wikipedia, « Compiler». En ligne. <https://en.wikipedia.org/wiki/Compiler>. Consulté le 2015-06-27

Wikipedia, « Integration testing ». En ligne. <https://en.wikipedia.org/wiki/Integration_testing>. Consulté le 2015-06-27

Krebs R, Momm C, Konev S (2012) Architectural concerns in multi-tenant SaaS applications. In: Proceedings of the 2nd international conference on cloud computing and service science (CLOSER'12). SciTePress. En ligne. <https://sdqweb.ipd.kit.edu/publications/pdfs/KrMoKo2012-closer-multitenant-sass.pdf>. Consulté le 2015-07-07

Internet Engineering Task Force (IETF), «The OAuth 2.0 Authorization Framework ». En ligne. <https://tools.ietf.org/html/rfc6749>. Consulté le 2015-07-16

Aaron Parecki (2012), « OAuth2 Simplified ». En ligne. <https://aaronparecki.com/articles/2012/07/29/1/oauth2-simplified>. Consulté le 2015-07-16

Flyway, « OAuth2 Simplified ». En ligne. <http://flywaydb.org/>. Consulté le 2015-07-30

ANNEXE I

SRS – SOFTWARE REQUIREMENTS SPECIFICATIONS

GTI_LOG_792_SRS-FRANCOEUR-DAVID.docx