



UNIVERSITÉ MOHAMMED V - RABAT  
ÉCOLE NATIONALE D'INFORMATIQUE ET D'ANALYSE DES SYSTÈMES

## RAPPORT DU PROJET SOA/MICROSERVICES

FILIÈRE

GÉNIE LOGICIEL

SUJET :

---

CONCEPTION ET DÉVELOPPEMENT D'UNE  
APPLICATION MICROSERVICES POUR UNE ENTREPRISE  
DE TRANSPORT URBAIN

---

*Réalisé par :*

BAKKALI Douae  
Meriem Abboud  
BEHRI Omar  
El Ammary Ilyas

*Encadré par :*  
Pr M. NASSAR

Année Universitaire 2025-2026

# Table des matières

<b>Introduction Générale</b>	<b>7</b>
<b>1 Présentation du projet</b>	<b>8</b>
1.1 Introduction . . . . .	8
1.2 Problématique . . . . .	8
1.2.1 Défis architecturaux . . . . .	8
1.2.2 Défis techniques et opérationnels . . . . .	8
1.2.3 Impact métier . . . . .	9
1.3 Solution proposée . . . . .	9
1.3.1 Services de domaine . . . . .	9
1.3.1.1 Services de gestion des utilisateurs et authentification . . . . .	9
1.3.1.2 Services de gestion des trajets et horaires . . . . .	9
1.3.1.3 Services de gestion des transactions et paiements . . . . .	9
1.3.1.4 Services de gestion de la mobilité et localisation . . . . .	10
1.3.1.5 Service de notifications . . . . .	10
1.3.2 Services transversaux et d'infrastructure . . . . .	10
1.3.3 Architecture technique . . . . .	10
1.4 Objectifs . . . . .	10
1.4.1 Objectifs pédagogiques . . . . .	10
1.4.2 Objectifs fonctionnels . . . . .	11
1.5 Gestion du projet . . . . .	12
1.5.1 Méthodologie . . . . .	12
1.5.2 Phases du projet . . . . .	12
1.6 Conclusion . . . . .	12
<b>2 Analyse fonctionnelle</b>	<b>13</b>
2.1 Introduction . . . . .	13
2.2 Acteurs du système . . . . .	13
2.3 Besoins fonctionnels . . . . .	13
2.3.1 Définition . . . . .	13
2.3.2 Besoins fonctionnels détaillés . . . . .	14
2.3.2.1 Authentification et gestion des utilisateurs . . . . .	14
2.3.2.2 Gestion des tickets . . . . .	14
2.3.2.3 Gestion des trajets et horaires . . . . .	14
2.3.2.4 Suivi géolocalisé des bus . . . . .	15

2.3.2.5	Gestion des abonnements . . . . .	15
2.3.2.6	Notifications . . . . .	15
2.4	Besoins non-fonctionnels . . . . .	15
2.4.1	Performance et résilience . . . . .	16
2.4.2	Sécurité . . . . .	16
2.4.3	Scalabilité et maintenabilité . . . . .	16
2.4.4	Compatibilité et accessibilité . . . . .	17
2.4.5	Conformité et légalité . . . . .	17
2.5	Diagramme global de cas d'utilisation . . . . .	17
2.5.1	Introduction . . . . .	17
2.5.2	Description détaillée des acteurs . . . . .	18
2.5.2.1	Passager . . . . .	18
2.5.2.2	Conducteur . . . . .	19
2.5.2.3	Administrateur . . . . .	19
2.5.3	Cas d'utilisation et relations . . . . .	19
2.5.3.1	Domaine Authentification . . . . .	19
2.5.3.2	Domaine Consultation et Recherche . . . . .	19
2.5.3.3	Domaine Transactions . . . . .	19
2.5.3.4	Domaine Gestion Métier . . . . .	20
2.5.4	Relations de dépendance . . . . .	20
2.5.4.1	Include (Dépendances obligatoires) . . . . .	20
2.5.4.2	Extend (Extensions optionnelles) . . . . .	20
2.6	Matrices de traçabilité . . . . .	20
2.6.1	Matrice Acteurs-Cas d'utilisation . . . . .	20
2.6.2	Matrice Cas d'utilisation-Services . . . . .	21
2.7	Conclusion . . . . .	21
<b>3</b>	<b>Architecture du système</b>	<b>22</b>
3.1	Introduction . . . . .	22
3.2	Architecture globale du système . . . . .	22
3.2.1	Vue d'ensemble et principes microservices . . . . .	22
3.2.2	Diagramme d'architecture du système . . . . .	23
3.2.2.1	Flux de communication . . . . .	23
3.3	Description des services . . . . .	23
3.3.1	Auth Service . . . . .	24
3.3.2	User Service . . . . .	24
3.3.3	Trip Service . . . . .	24
3.3.4	Ticket Service . . . . .	24
3.3.5	Subscription Service . . . . .	24
3.3.6	Geolocation Service . . . . .	25
3.3.7	Payment Service . . . . .	25
3.3.8	Notification Service . . . . .	25
3.3.9	API Gateway . . . . .	25

3.4	Communication et intégration . . . . .	25
3.4.1	Communication inter-services (REST et Kafka) . . . . .	25
3.4.1.1	Communication Synchrone (REST/HTTP) . . . . .	25
3.4.1.2	Communication Asynchrone (Apache Kafka) . . . . .	26
3.4.2	Gestion des erreurs et tolérance aux pannes . . . . .	26
3.5	Sécurité et authentification . . . . .	26
3.5.1	Mécanismes d'authentification et d'autorisation . . . . .	26
3.5.2	Gestion des secrets . . . . .	26
3.6	Infrastructure et déploiement . . . . .	27
3.6.1	Conteneurisation et orchestration (Docker, Kubernetes) . . . . .	27
3.6.2	Base de données décentralisée (Database per Service) . . . . .	27
3.6.3	Monitoring et logs (Observabilité) . . . . .	27
3.7	Conclusion . . . . .	27
<b>4</b>	<b>Conception détaillée</b>	<b>28</b>
4.1	Introduction . . . . .	28
4.2	Conception par service . . . . .	28
4.2.1	Service Authentification (Auth Service) . . . . .	28
4.2.1.1	Objectif . . . . .	28
4.2.1.2	Cas d'utilisation . . . . .	28
4.2.1.3	Diagramme de classes . . . . .	29
4.2.1.4	Base de données et entités (Database per Service) . . . . .	30
4.2.1.5	API REST / Endpoints . . . . .	30
4.2.1.6	Diagramme de séquence – Connexion utilisateur . . . . .	31
4.2.2	Service Gestion des Utilisateurs (User Service) . . . . .	31
4.2.2.1	Objectif . . . . .	31
4.2.2.2	Cas d'utilisation . . . . .	31
4.2.2.3	Diagramme de classes . . . . .	32
4.2.2.4	Base de données et entités (Database per Service) . . . . .	33
4.2.2.5	API REST / Endpoints . . . . .	33
4.2.2.6	Diagramme de séquence – Inscription utilisateur . . . . .	34
4.2.3	Service Achat de Tickets (Ticket Service) . . . . .	34
4.2.3.1	Objectif . . . . .	34
4.2.3.2	Cas d'utilisation . . . . .	34
4.2.3.3	Diagramme de classes . . . . .	35
4.2.3.4	Base de données et entités (Database per Service) . . . . .	36
4.2.3.5	API REST / Endpoints . . . . .	36
4.2.3.6	Diagramme de séquence – Achat d'un ticket . . . . .	37
4.2.3.7	Interactions avec les autres services . . . . .	37
4.2.3.8	Résumé du flux global . . . . .	37
4.2.4	Service Gestion des Abonnements (Subscription Service) . . . . .	38
4.2.4.1	Objectif . . . . .	38
4.2.4.2	Cas d'utilisation . . . . .	38

4.2.4.3	Diagramme de classes . . . . .	38
4.2.4.4	Base de données et entités (Database per Service) . . . . .	39
4.2.4.5	API REST / Endpoints . . . . .	40
4.2.4.6	Diagramme de séquence – Renouvellement d'un abonnement . . . . .	40
4.2.5	Service Gestion des Trajets et Horaires (Trip Service) . . . . .	40
4.2.5.1	Objectif . . . . .	40
4.2.5.2	Cas d'utilisation . . . . .	40
4.2.5.3	Diagramme de classes . . . . .	41
4.2.5.4	Base de données et entités (Database per Service) . . . . .	44
4.2.5.5	API REST / Endpoints . . . . .	45
4.2.5.6	Diagramme de séquence – Recherche de trajet par passager . . . . .	46
4.2.6	Service Géolocalisation des Bus (Geolocation Service) . . . . .	46
4.2.6.1	Objectif . . . . .	46
4.2.6.2	Cas d'utilisation . . . . .	46
4.2.6.3	Diagramme de classes . . . . .	47
4.2.6.4	Base de données et entités (Database per Service) . . . . .	49
4.2.6.5	API REST / Endpoints . . . . .	49
4.2.6.6	Diagramme de séquence – Suivi en temps réel d'un bus . . . . .	50
4.2.7	Service Notifications (Notification Service) . . . . .	51
4.2.7.1	Objectif . . . . .	51
4.2.7.2	Cas d'utilisation . . . . .	51
4.2.7.3	Diagramme de classes . . . . .	52
4.2.7.4	Base de données et entités (Database per Service) . . . . .	54
4.2.7.5	API REST / Endpoints . . . . .	54
4.2.7.6	Diagramme de séquence – Envoi de notification de retard . . . . .	56
4.2.8	Service Payment (Payment Service) . . . . .	56
4.2.8.1	Objectif . . . . .	56
4.2.8.2	Cas d'utilisation . . . . .	57
4.2.8.3	Diagramme de classes . . . . .	58
4.2.8.4	Base de données et entités (Database per Service) . . . . .	60
4.2.8.5	API REST / Endpoints . . . . .	60
4.2.8.6	Diagramme de séquence – Achat d'un ticket . . . . .	62
4.3	Conclusion . . . . .	63

# Table des figures

2.1	Diagramme global de cas d'utilisation - Vue consolidée du système . . . . .	18
3.1	Architecture globale du système de Transport Urbain . . . . .	23
4.1	Diagramme de cas d'utilisation du service Authentification . . . . .	29
4.2	Diagramme de classes du service Authentification . . . . .	29
4.3	Architecture de la base de données pour le service Authentification . . . . .	30
4.4	Diagramme de séquence du processus de connexion utilisateur . . . . .	31
4.5	Diagramme de cas d'utilisation du service Gestion des Utilisateurs . . . . .	32
4.6	Diagramme de classes du service Gestion des Utilisateurs . . . . .	33
4.7	Architecture de la base de données pour le service Gestion des Utilisateurs . . . . .	33
4.8	Diagramme de séquence du processus d'inscription utilisateur . . . . .	34
4.9	Diagramme de cas d'utilisation du service d'achat de tickets . . . . .	35
4.10	Diagramme de classes du service d'achat de tickets . . . . .	35
4.11	Schéma de la base de données pour le service d'achat de tickets . . . . .	36
4.12	Diagramme de séquence du processus d'achat de ticket . . . . .	37
4.13	Diagramme de cas d'utilisation du service Gestion des Abonnements . . . . .	38
4.14	Diagramme de classes du service Gestion des Abonnements . . . . .	39
4.15	Architecture de la base de données pour le service Gestion des Abonnements . . . . .	39
4.16	Diagramme de séquence du processus de renouvellement d'un abonnement . . . . .	40
4.17	Diagramme de cas d'utilisation du service Gestion des Trajets et Horaires . . . . .	41
4.18	Diagramme de classes du service Gestion des Trajets et Horaires . . . . .	43
4.19	Architecture de la base de données pour le service Gestion des Trajets . . . . .	44
4.20	Diagramme de séquence du processus de recherche de trajet . . . . .	46
4.21	Diagramme de cas d'utilisation du service Géolocalisation . . . . .	47
4.22	Diagramme de classes du service Géolocalisation . . . . .	48
4.23	Architecture de la base de données pour le service Géolocalisation . . . . .	49
4.24	Diagramme de séquence du processus de tracking en temps réel . . . . .	50
4.25	Diagramme de cas d'utilisation du service Notifications . . . . .	52
4.26	Diagramme de classes du service Notifications . . . . .	53
4.27	Architecture de la base de données pour le service Notifications . . . . .	54
4.28	Diagramme de séquence du processus d'envoi de notification de retard . . . . .	56
4.29	Diagramme de cas d'utilisation du service Payment . . . . .	58
4.30	Diagramme de classes du service Payment . . . . .	59
4.31	Architecture de la base de données pour le service Payment . . . . .	60
4.32	Diagramme de séquence du processus d'achat de ticket avec paiement . . . . .	62

# Introduction Générale

Le transport urbain constitue un enjeu stratégique majeur pour les villes modernes. Avec l'augmentation croissante de la population urbaine et les défis liés à la mobilité durable, les entreprises de transport doivent moderniser leurs services pour offrir une expérience utilisateur optimale et une gestion opérationnelle efficace.

Ce projet a pour objectif de concevoir, développer et déployer un **système de gestion intégré pour une entreprise de transport urbain** basé sur une architecture microservices. Cette approche architectural innovante permet une amélioration significative des services existants en offrant une scalabilité granulaire, une maintenance simplifiée et une capacité d'évolution continue.

À travers ce projet, nous serons en mesure de :

- **Acquérir une maîtrise approfondie de l'architecture microservices** en identifiant les composants clés et en comprenant leurs interactions complexes, permettant ainsi une décomposition intelligente des fonctionnalités métier.
- **Maîtriser la conception d'API RESTful robustes et sécurisées** servant de contrats d'interfaçage entre les services, garantissant l'interopérabilité et la réutilisabilité du code.
- **Implémenter des mécanismes de communication asynchrone** via des brokers de messages (Kafka) et synchrone via des API Gateways, optimisant ainsi la résilience du système.
- **Gérer des bases de données distribuées et décentralisées** selon le paradigme "Database per Service", assurant l'autonomie et l'indépendance de chaque microservice.
- **Orchestrer le déploiement et la scalabilité** des applications conteneurisées via Docker et Kubernetes sur une infrastructure cloud moderne (Google Cloud Platform).
- **Développer une interface utilisateur intuitive et responsive** offrant une expérience utilisateur fluide et accessible sur tous les appareils.

Le système développé permettra aux utilisateurs finaux de bénéficier de services modernes : consultation des horaires en temps réel, achat de tickets simplifiés, suivi géolocalisé des véhicules, et gestion flexible des abonnements. Cette solution représente une transformation digitale complète du secteur du transport urbain.

# Chapitre 1

## Présentation du projet

### 1.1 Introduction

Ce chapitre présente le contexte général du projet, les défis identifiés dans le domaine du transport urbain, et la solution proposée pour y répondre. Il établit également les objectifs pédagogiques et fonctionnels ainsi que les méthodes de gestion du projet.

### 1.2 Problématique

Le secteur du transport urbain fait face à des défis technologiques et organisationnels majeurs. Les systèmes informatiques actuels, souvent basés sur une architecture monolithique, présentent plusieurs limitations critiques :

#### 1.2.1 Défis architecturaux

- **Rigidité architecturale** : Les applications monolithiques rendent difficile l'évolution indépendante des fonctionnalités. Toute modification d'un composant requiert de redéployer l'ensemble du système, limitant l'agilité de l'entreprise et sa capacité d'adaptation aux changements du marché.
- **Déploiements complexes et risqués** : Les mises à jour impliquent des fenêtres de maintenance étendues et des risques de régression élevés. Un bug mineur peut entraîner l'indisponibilité complète du service, impactant directement la qualité du service aux utilisateurs finaux.
- **Scalabilité limitée** : Il est impossible de scaler sélectivement les fonctionnalités critiques. Si un module requiert une puissance de calcul accrue, l'ensemble de l'application doit être dupliquée, entraînant un gaspillage de ressources et des coûts d'infrastructure disproportionnés.

#### 1.2.2 Défis techniques et opérationnels

- **Dépendances techniques fortement couplées** : Un bug ou une défaillance dans un module affecte potentiellement l'ensemble du système. Le couplage fort entre les composants crée des points de défaillance uniques (Single Point of Failure).
- **Maintenance et évolution coûteuses** : L'onboarding de nouveaux développeurs est complexe en raison de la base de code monolithique volumineux. Les cycles de développement sont ralentis, et la

productivité des équipes diminue proportionnellement à la croissance du projet.

- **Absence d'indépendance technologique** : Tous les modules utilisent obligatoirement les mêmes technologies (langage, frameworks, bases de données), limitant les choix technologiques optimaux pour chaque fonctionnalité spécifique.

### 1.2.3 Impact métier

Ces problèmes se traduisent par : une réactivité commerciale réduite face à la concurrence, une qualité de service dégradée due aux fréquentes indisponibilités, et une expérience utilisateur insatisfaisante comparée aux solutions modernes disponibles sur le marché.

## 1.3 Solution proposée

Pour remédier à ces défis, nous proposons l'implémentation d'une **architecture microservices complète et distribuée**. Cette approche consiste à décomposer l'application en services autonomes, indépendants et spécialisés, communiquant via des interfaces bien définies.

### 1.3.1 Services de domaine

L'architecture proposée comprend les services métier suivants, organisés autour des domaines métier clés :

#### 1.3.1.1 Services de gestion des utilisateurs et authentification

- **Service d'Authentification** : Gère l'authentification centralisée des utilisateurs et l'émission des tokens JWT pour sécuriser l'accès aux autres services. Responsable de la validation des identifiants et du contrôle d'accès basé sur les rôles.
- **Service Utilisateurs** : Maintient les profils, informations personnelles et préférences des utilisateurs (passagers, conducteurs, administrateurs). Gère les données de compte, les méthodes de paiement préférées et les paramètres de notification.

#### 1.3.1.2 Services de gestion des trajets et horaires

- **Service Trajets** : Fournit les informations détaillées sur les horaires, trajets, arrêts et itinéraires des bus. Gère également l'affectation des conducteurs et véhicules aux trajets.
- **Service de Planification (Scheduling Service)** : Automatise la création et l'ajustement des horaires et affectations des bus/conducteurs en fonction du trafic et des données historiques.

#### 1.3.1.3 Services de gestion des transactions et paiements

- **Service Tickets** : Permet l'achat, la gestion et la validation des tickets de transport, offrant diverses options de tarification et de durée de validité. Gère le cycle de vie complet des tickets (création, validation, remboursement).
- **Service Abonnements** : Gère les formules d'abonnement flexibles (mensuels, annuels, spéciaux) avec tarification progressive et avantages associés. Orchestre les renouvellements automatiques et les résiliations.

- **Service Paiement** : Traite les transactions financières de manière sécurisée, gère les portefeuilles numériques, les remboursements et assure la conformité PCI-DSS. Intègre les passerelles de paiement externes (cartes bancaires, portefeuilles numériques).

#### 1.3.1.4 Services de gestion de la mobilité et localisation

- **Service Géolocalisation** : Suit en temps réel la position des véhicules et calcule les estimations d'arrivée via intégration GPS et APIs cartographiques (Google Maps, OpenStreetMap). Détecte les retards et génère les événements de mise à jour de localisation.

#### 1.3.1.5 Service de notifications

- **Service Notifications** : Envoie des alertes proactives aux utilisateurs (retards, annulations, mises à jour de tarifs, confirmations de transactions) via email et SMS. Gère les préférences de notification par canal et par type d'alerte.

### 1.3.2 Services transversaux et d'infrastructure

- **API Gateway** : Point d'entrée unique centralisant l'accès aux microservices, assurant le routage, l'authentification et le rate-limiting.
- **Service de Configuration (Config Service)** : Gestion centralisée et dynamique des configurations applicatives sans redéploiement.
- **Service de Découverte (Service Discovery)** : Enregistrement automatique et découverte dynamique des services dans l'infrastructure distribuée.
- **Service de Monitoring (Monitoring Service)** : Collecte centralisée des logs, métriques de performance et alertes système pour la supervision opérationnelle.

### 1.3.3 Architecture technique

- **Base de données décentralisée** : Chaque microservice dispose de sa propre instance de base de données (PostgreSQL), garantissant l'isolation complète des données et l'indépendance d'évolution.
- **Communication inter-services** : Combinaison de communication synchrone (REST/HTTP) pour les opérations critiques et asynchrone (Kafka) pour l'échange d'événements métier.
- **Conteneurisation** : Chaque service est package dans un conteneur Docker, assurant la consistance entre les environnements de développement, test et production.
- **Orchestration** : Kubernetes gère le déploiement, la scalabilité automatique et la résilience des conteneurs en environnement production.

## 1.4 Objectifs

### 1.4.1 Objectifs pédagogiques

Les objectifs pédagogiques visent à développer les compétences techniques et méthodologiques essentielles pour le développement logiciel moderne :

- **Comprendre et maîtriser l'architecture microservices** : Analyser les principes SOLID appliqués à l'architecture distribuée, identifier les patterns appropriés (Saga, Circuit Breaker, Event Sourcing) et évaluer les trade-offs entre complexité et bénéfices.
- **Décomposer une application monolithique** : Appliquer les techniques de Domain-Driven Design (DDD) pour identifier les domaines métier, définir les bounded contexts et délimiter les responsabilités de chaque service.
- **Concevoir des APIs RESTful robustes** : Maîtriser les conventions HTTP, implémenter la versioning d'API, gérer les erreurs de manière cohérente et documenter les API selon la spécification OpenAPI/Swagger.
- **Implémenter des patterns de communication** : Comparer les approches synchrone et asynchrone, implémenter des patterns comme Saga distribué et événements pour maintenir la cohérence transactionnelle.
- **Gérer la persistance des données distribuées** : Appliquer le pattern "Database per Service", gérer les migrations distribuées et assurer la cohérence logique sans dépendances de base de données directes.
- **Orchestrer le déploiement cloud** : Maîtriser les conteneurs Docker, concevoir des deployments Kubernetes résilients avec stratégies de rolling updates et auto-scaling.
- **Développer l'expérience utilisateur** : Créer des interfaces modernes, responsive et performantes avec les dernières technologies frontend (React, Next.js).

#### 1.4.2 Objectifs fonctionnels

Les objectifs fonctionnels définissent les capacités concrètes que le système doit fournir aux utilisateurs finaux :

- **Consultation en temps réel** : Permettre aux utilisateurs de consulter les horaires des bus, les trajets disponibles et les estimations d'arrivée avec une latence minimale.
- **Achat simplifié de tickets** : Offrir une procédure d'achat fluide avec support de multiples moyens de paiement, génération instantanée de tickets numériques et validation mobile.
- **Suivi géolocalisé** : Afficher en temps réel la localisation des véhicules sur une carte interactive, permettant aux utilisateurs de planifier leurs trajets avec précision.
- **Gestion flexible des abonnements** : Proposer des plans d'abonnement variés (mensuels, annuels, spéciaux) avec calcul automatique des tarifs et gestion des renouvellements.
- **Notifications proactives** : Alerter les utilisateurs automatiquement sur les changements importants (retards, annulations, mises à jour tarifaires).
- **Sécurité et confidentialité** : Protéger les données sensibles avec authentification robuste, chiffrement en transit et au repos, et conformité réglementaire.
- **Performance et disponibilité** : Assurer un temps de réponse < 500ms pour 95% des requêtes et une disponibilité du service de 99.5% minimum.

## 1.5 Gestion du projet

### 1.5.1 Méthodologie

Ce projet sera conduit selon une **approche Agile**, spécifiquement une adaptation de Scrum pour un contexte d'équipe réduite. Cette méthodologie offre plusieurs avantages :

- **Itérations courtes** : Cycles de développement de 2 semaines permettant une intégration rapide du feedback et une correction précoce des dérives.
- **Transparence** : Réunions quotidiennes de synchronisation (daily standups) assurant l'alignement de l'équipe et l'identification précoce des obstacles.
- **Livraison progressive** : Incrément de fonctionnalités complètes et testées à chaque fin de sprint, minimisant les risques d'intégration de dernier moment.
- **Adaptation** : Flexibilité pour ajuster les priorités en fonction de l'apprentissage et de l'évolution des besoins.

### 1.5.2 Phases du projet

Le projet est structuré en cinq phases majeures :

1. **Phase 1 - Conception (2 semaines)** : Modélisation architecturale, définition des services, conception des API et des bases de données.
2. **Phase 2 - Implémentation (8 semaines)** : Développement itératif de chaque microservice en parallèle, avec intégration continue.
3. **Phase 3 - Test (2 semaines)** : Tests unitaires, d'intégration, de performance et tests de chaos pour valider la résilience.
4. **Phase 4 - Déploiement (1 semaine)** : Déploiement en environnement production sur Google Cloud avec Kubernetes et monitoring.
5. **Phase 5 - Documentation (1 semaine)** : Documentation des API, guides d'administration et documentation d'architecture.

## 1.6 Conclusion

Ce projet représente une expérience complète du cycle de vie d'un système distribué complexe. À travers la conception, le développement et le déploiement d'une solution microservices pour le transport urbain, nous acquerrons les compétences essentielles exigées par le marché du développement logiciel moderne. L'application pratique des principes d'architecture distribuée, associée à la maîtrise des outils cloud natifs, prépare à relever les défis technologiques des systèmes scalables et résilients.

# Chapitre 2

## Analyse fonctionnelle

### 2.1 Introduction

L'analyse fonctionnelle constitue une étape fondamentale du cycle de développement logiciel. Elle consiste à identifier, documenter et valider les exigences que le système doit satisfaire pour répondre aux besoins des utilisateurs. Ce chapitre présente les acteurs impliqués dans le système, détaille les besoins fonctionnels et non-fonctionnels, et illustre les interactions principales par un diagramme global de cas d'utilisation.

### 2.2 Acteurs du système

Un acteur représente une entité externe (utilisateur ou système) qui interagit avec l'application. Pour notre système de transport urbain, nous identifions les acteurs suivants :

- **Passager** : Utilisateur final qui consulte les horaires, achète des tickets, suit la localisation des bus et gère ses abonnements.
- **Conducteur** : Opérateur du véhicule responsable de la conformité des trajets, de la mise à jour en temps réel de la localisation et de la gestion des incidents.
- **Administrateur** : Responsable de la gestion des trajets, des horaires, des tarifs et de la définition des abonnements.

### 2.3 Besoins fonctionnels

#### 2.3.1 Définition

Un **besoin fonctionnel** est une exigence spécifique qui décrit précisément **ce que le système doit faire** pour satisfaire les objectifs métier et les attentes des utilisateurs. Il se concentre sur les fonctionnalités, les interactions et les services que l'application doit fournir. Les besoins fonctionnels répondent à la question fondamentale : "Quelles actions le système doit-il permettre ?"

Les besoins fonctionnels sont généralement exprimés sous plusieurs formes complémentaires :

- **Cas d'utilisation** : Séquences d'actions entre l'acteur et le système décrivant un scénario métier complet.

- **Scénarios** : Descriptions détaillées de situations réelles incluant les chemins heureux et les cas d'exception.
- **Spécifications détaillées** : Documentation des règles métier, des validations, des calculs et des contraintes logiques.

### 2.3.2 Besoins fonctionnels détaillés

Les besoins fonctionnels du système sont organisés par domaine métier :

#### 2.3.2.1 Authentification et gestion des utilisateurs

- **S'authentifier** : L'utilisateur doit pouvoir se connecter via identifiant/mot de passe ou authentification fédérée (OAuth 2.0), recevoir un token JWT valide pour 24h.
- **S'inscrire** : Enregistrement de nouveaux passagers avec validation des données (email unique, numéro de téléphone, adresse), envoi d'email de confirmation.
- **Réinitialiser le mot de passe** : Procédure sécurisée de réinitialisation via email, lien valide 1h, création automatique du nouveau mot de passe.
- **Gérer son profil** : Consultation et modification des informations personnelles, préférences de notification, méthodes de paiement préférées.
- **Consulter l'historique des transactions** : Accès à tous les achats, abonnements et transactions avec détail complet et reçus.

#### 2.3.2.2 Gestion des tickets

- **Rechercher des trajets** : L'utilisateur spécifie une origine, une destination et une date/heure ; le système retourne les trajets disponibles avec horaires et tarifs.
- **Acheter un ticket** : Sélection du trajet, variante du ticket (enfant, senior, réduit), validation du panier, paiement sécurisé, génération d'un QR code valide.
- **Consulter les tickets** : Accès à la liste des tickets achetés avec statuts (valide, expiré, utilisé), détails complets et téléchargement en PDF.
- **Valider un ticket** : Le conducteur scanne le QR code à bord du bus pour confirmer l'utilisation du ticket et générer un reçu.
- **Annuler un ticket** : Remboursement intégral si annulation > 2h avant le départ, remboursement partiel sinon selon tarification.

#### 2.3.2.3 Gestion des trajets et horaires

- **Consulter les horaires** : Affichage des horaires de tous les trajets disponibles, avec jours d'exploitation et prochains départs.
- **Ajouter un trajet** : L'Administrateur crée un nouveau trajet avec arrêts, durée estimée, capacité du bus, périodicité.
- **Modifier un horaire** : Mise à jour des horaires existants avec notification automatique des passagers abonnés.

- **Consulter les arrêts** : Affichage détaillé de tous les arrêts d'une ligne incluant adresse, accessibilité PMR, équipements.
- **Publier les avis de perturbation** : Annonces de changements exceptionnels (grèves, travaux) avec notification aux utilisateurs affectés.

#### 2.3.2.4 Suivi géolocalisé des bus

- **Afficher la localisation en temps réel** : Visualisation de la position actuelle du bus sur la carte avec estimation d'arrivée mise à jour chaque 30 secondes.
- **Calculer l'ETA** : Estimation dynamique du temps d'arrivée aux prochains arrêts basée sur GPS, conditions de trafic et historique.
- **Suivre un bus** : L'utilisateur peut suivre un bus spécifique et reçoit des notifications aux étapes clés du trajet.
- **Gérer les retards** : Détection automatique des retards > 5 min avec alertes proactives aux passagers et équipes opérationnelles.

#### 2.3.2.5 Gestion des abonnements

- **Consulter les offres** : Affichage des plans d'abonnement disponibles (mensuel, annuel, spécial) avec tarifs et avantages associés.
- **Souscrire un abonnement** : Création d'abonnement avec durée choisie, paiement automatique du montant correspondant.
- **Renouveler l'abonnement** : Renouvellement automatique ou manuel avec prélèvement automatique selon les préférences.
- **Consulter les détails** : Affichage du statut (actif, expiré, suspendu), date d'expiration, trajets inclus et solde de tickets.
- **Annuler un abonnement** : Résiliation avec remboursement au prorata du solde non utilisé.

#### 2.3.2.6 Notifications

- **Notifier les retards** : Envoi automatique d'email/SMS à tous les passagers du trajet affecté lors d'un retard > 5 minutes.
- **Confirmer les achats** : Confirmation de chaque transaction par email avec reçu et détails du ticket.
- **Alerter sur les mises à jour** : Notifications sur les changements d'horaire, annulations de trajets ou offres promotionnelles.
- **Configurer les préférences** : Les utilisateurs peuvent activer/désactiver les notifications par canal (email, SMS) et par type.

### 2.4 Besoins non-fonctionnels

Les **besoins non-fonctionnels** définissent les critères de qualité et les contraintes techniques que le système doit respecter. Ils répondent à la question : "Comment le système doit-il fonctionner ?"

#### 2.4.1 Performance et résilience

- **Temps de réponse** : 95% des requêtes doivent être traitées en moins de 500ms, 99% en moins de 2 secondes.
- **Disponibilité** : Le système doit garantir une disponibilité de 99.5% par mois (downtime autorisé : 3.6 heures/mois).
- **Débit** : Supporter au minimum 10 000 requêtes concurrentes sans dégradation de service.
- **Récupération après défaillance** : RTO (Recovery Time Objective) < 5 minutes, RPO (Recovery Point Objective) < 1 minute.
- **Résilience** : Tolérer la défaillance d'un service sans impact sur les autres, implémentation de patterns Circuit Breaker et Retry exponentiels.

#### 2.4.2 Sécurité

- **Authentification** : Toutes les requêtes doivent être authentifiées via JWT ou session validée, tokens possédant une durée de vie limitée.
- **Autorisation** : Implémentation de contrôle d'accès basé sur les rôles (RBAC) avec principes du moindre privilège.
- **Confidentialité des données** : Toutes les données en transit doivent être chiffrées (HTTPS/TLS 1.3 minimum), données sensibles chiffrées en base de données.
- **Protection des paiements** : Conformité PCI-DSS, pas de stockage direct des numéros de carte, utilisation de tokens de paiement.
- **Audit et logging** : Journalisation de toutes les opérations sensibles (authentification, transactions) avec traçabilité complète.
- **Gestion des secrets** : Rotation des credentials, utilisation de gestionnaire de secrets centralisé (Vault), aucun secret en dur dans le code.

#### 2.4.3 Scalabilité et maintenabilité

- **Scalabilité horizontale** : Chaque service doit pouvoir être déployé en multiples instances pour absorber les pics de charge.
- **Évolutivité du code** : Architecture modulaire permettant l'ajout de nouvelles fonctionnalités sans modifications majeures des services existants.
- **Maintenabilité** : Code documenté, testabilité > 80%, respect des standards de codage, versioning sémantique des APIs.
- **Monitoring** : Collecte de métriques (CPU, mémoire, latence, erreurs), alertes automatiques sur les seuils d'anomalies.
- **Observabilité** : Logs structurés (JSON), traces distribuées permettant de suivre une requête dans tout le système, sampling des métriques.

#### 2.4.4 Compatibilité et accessibilité

- **Compatibilité navigateur** : Support des navigateurs modernes (Chrome, Firefox, Safari, Edge) et de leurs versions courantes (-2 versions).
- **Compatibilité mobile** : Application responsive optimisée pour smartphones (iOS, Android) et tablettes.
- **Accessibilité** : Conformité WCAG 2.1 niveau AA minimum, support des lecteurs d'écran, navigation au clavier complète.
- **Internationalisation** : Support du français et de l'anglais, gestion des fuseaux horaires, format des dates/devises adaptés.

#### 2.4.5 Conformité et légalité

- **Protection des données** : Conformité RGPD (droit d'accès, effacement, portabilité), politique de confidentialité, consentement utilisateur.
- **Traçabilité** : Audit trail complet des transactions pour conformité réglementaire et résolution des litiges.
- **Sauvegardes** : Sauvegardes quotidiennes avec test de restauration mensuel, géo-redondance des backups.

### 2.5 Diagramme global de cas d'utilisation

#### 2.5.1 Introduction

Le diagramme de cas d'utilisation constitue une représentation graphique synthétique des interactions entre les acteurs (utilisateurs et systèmes) et le système. Il modélise les fonctionnalités principales du système de transport urbain en mettant en évidence :

- Les quatre catégories d'acteurs et leurs rôles respectifs
- Les cas d'utilisation (fonctionnalités) disponibles pour chaque acteur
- Les relations de dépendance entre les cas d'utilisation (includes et extends)
- Les limites du système (boundaries)

Ce diagramme offre une vue consolidée avant la décomposition détaillée par domaine métier dans les chapitres suivants.

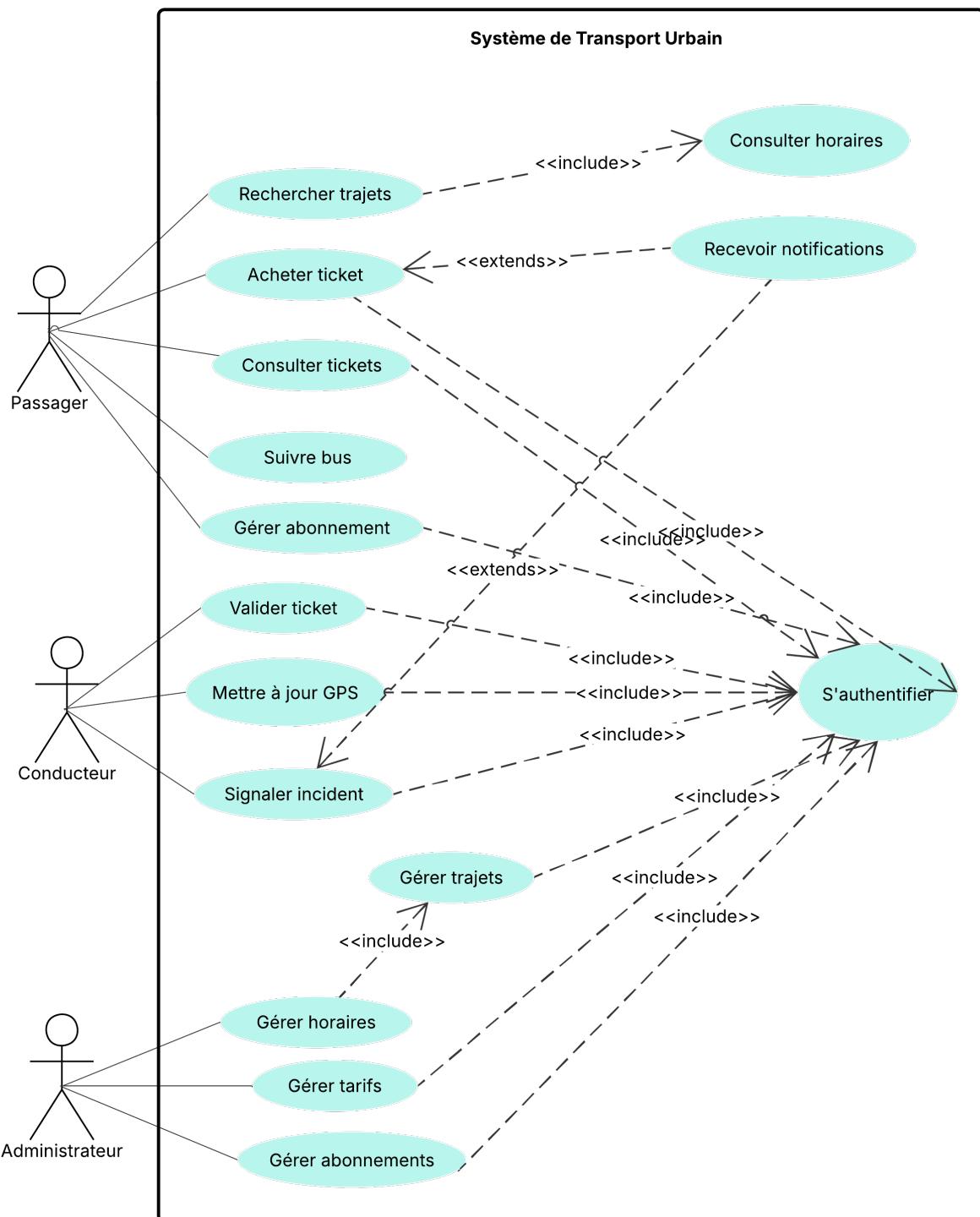


FIGURE 2.1 – Diagramme global de cas d'utilisation - Vue consolidée du système

## 2.5.2 Description détaillée des acteurs

### 2.5.2.1 Passager

Le passager représente l'utilisateur final du système. Ses interactions principales incluent :

- **S'authentifier** : Accès sécurisé au système via identifiants
- **Consulter horaires** : Consultation de la grille horaire des bus disponibles
- **Rechercher trajets** : Recherche de trajets entre deux points avec sélection de date/heure
- **Acheter ticket** : Acquisition de tickets individuels avec paiement
- **Consulter tickets** : Accès à l'historique et aux tickets validés
- **Suivre bus** : Suivi en temps réel de la position et ETA du véhicule
- **Gérer abonnement** : Souscription, renouvellement et résiliation d'abonnements
- **Recevoir notifications** : Alertes automatiques sur retards, changements, confirmations

### 2.5.2.2 Conducteur

Le conducteur est l'opérateur du véhicule responsable de son exploitation. Ses interactions comprennent :

- **S'authentifier** : Identification sécurisée du conducteur (traçabilité)
- **Valider ticket** : Vérification des titres de transport des passagers via scan QR
- **Mettre à jour GPS** : Transmission en continu de la position du bus
- **Signaler incident** : Notification des problèmes opérationnels (accidents, pannes, retards)

### 2.5.2.3 Administrateur

L'Administrateur gère les données et les règles commerciales du système. Ses interventions portent sur :

- **S'authentifier** : Accès avec privilèges de gestion métier
- **Gérer trajets** : Création et modification des lignes de bus, arrêts, itinéraires
- **Gérer horaires** : Configuration des grilles horaires par trajet et jour de semaine
- **Gérer tarifs** : Définition des prix des tickets et tarifications spéciales
- **Gérer abonnements** : Création et modification des formules d'abonnement disponibles

## 2.5.3 Cas d'utilisation et relations

### 2.5.3.1 Domaine Authentication

**S'authentifier** est le cas d'utilisation central du système. Tous les cas sensibles l'incluent obligatoirement pour garantir la traçabilité, la sécurité et l'isolement des données personnelles.

### 2.5.3.2 Domaine Consultation et Recherche

**Consulter horaires** fournit la grille horaire du système. Ce cas est fondamental car :

- **Rechercher trajets «include» Consulter horaires** : Pour chercher un trajet, le système doit afficher les horaires pour permettre au passager de sélectionner une date et obtenir les trajets disponibles.

### 2.5.3.3 Domaine Transactions

**Acheter ticket** combine plusieurs fonctionnalités : authentication obligatoire, sélection du trajet et de l'horaire, traitement du paiement sécurisé, génération du ticket numérique avec code QR.

### 2.5.3.4 Domaine Gestion Métier

Les cas **Gérer horaires**, **Gérer tarifs** et **Gérer abonnements** incluent tous **Gérer trajets** car :

- **Gérer horaires «include» Gérer trajets** : Les horaires appartiennent à des trajets spécifiques. Il faut d'abord sélectionner le trajet avant de modifier ses horaires. Cette dépendance garantit l'intégrité des données métier.

### 2.5.4 Relations de dépendance

#### 2.5.4.1 Include (Dépendances obligatoires)

Une relation «include» exprime une dépendance **obligatoire**.

**Exemples principaux :**

- **Acheter ticket «include» S'authentifier** : Impossible d'acheter sans se connecter
- **Rechercher trajets «include» Consulter horaires** : La recherche doit afficher les horaires disponibles
- **Gérer horaires «include» Gérer trajets** : Les horaires appartiennent à un trajet, sélection obligatoire
- **S'authentifier** : Incluse par 12 autres cas sensibles

#### 2.5.4.2 Extend (Extensions optionnelles)

Une relation «extend» représente une fonctionnalité **optionnelle**.

- **Recevoir notifications «extend» Acheter ticket** : Notification optionnelle après achat
- **Recevoir notifications «extend» Signaler incident** : Notification optionnelle lors d'incident

## 2.6 Matrices de traçabilité

### 2.6.1 Matrice Acteurs-Cas d'utilisation

Cas d'utilisation	Passager	Conducteur	Administrateur	
S'authentifier	✓	✓	✓	
Consulter horaires	✓			
Rechercher trajets	✓			
Acheter ticket	✓			
Suivre bus	✓			
Gérer abonnement	✓			
Valider ticket		✓		
Signaler incident		✓		
Mettre à jour GPS		✓		
Gérer trajets			✓	
Gérer horaires			✓	
Gérer tarifs			✓	
Gérer abonnements			✓	

TABLE 2.1 – Matrice Acteurs-Cas d'utilisation

## 2.6.2 Matrice Cas d'utilisation-Services

Cas d'utilisation	Auth	User	Ticket	Journey	Geo	Sub	Paiement	Notif
S'authentifier	✓							
Consulter horaires				✓				
Rechercher trajets				✓				
Acheter ticket	✓		✓	✓			✓	✓
Suivre bus					✓			
Gérer abonnement	✓	✓		✓		✓	✓	✓
Consulter tickets			✓					
Valider ticket			✓					
Signaler incident					✓			✓
Mettre à jour GPS					✓			

TABLE 2.2 – Matrice Cas d'utilisation-Services microservices

## 2.7 Conclusion

L'analyse fonctionnelle présentée dans ce chapitre établit les fondations nécessaires pour une architecture microservices cohérente et bien structurée. L'identification précise des acteurs, la définition détaillée des besoins fonctionnels et non-fonctionnels, ainsi que la visualisation des cas d'utilisation constituent un référentiel commun pour toutes les phases ultérieures du projet.

Les matrices de traçabilité assurent une couverture complète des exigences, évitant ainsi les oubliés ou les dérives de scope. Les besoins non-fonctionnels notamment—performance, sécurité, scalabilité—guideront les choix architecturaux et technologiques détaillés dans le chapitre suivant de conception.

Ce document d'analyse sert également de validation avec les stakeholders métier et techniques, garantissant l'alignement du projet avec les attentes réelles de l'organisation.

# Chapitre 3

## Architecture du système

### 3.1 Introduction

Ce chapitre détaille l'architecture technique retenue pour le système de transport urbain. Suite à l'analyse fonctionnelle, nous avons opté pour une **architecture microservices** afin de garantir la scalabilité, la résilience et l'agilité requises par un système temps réel et transactionnel. Nous présentons la vue d'ensemble, les principes clés, la description de chaque microservice, les mécanismes de communication inter-services, et les choix d'infrastructure pour le déploiement.

### 3.2 Architecture globale du système

#### 3.2.1 Vue d'ensemble et principes microservices

L'architecture globale est basée sur le pattern **Microservices** orchestré par des technologies Cloud Native. Ce choix permet de décomposer le système en petites unités de services autonomes, chacun gérant un domaine métier spécifique (selon le principe du Domain-Driven Design ou *DDD*).

Les principes fondamentaux appliqués sont :

- **Découplage** : Chaque service est autonome, indépendant dans son cycle de vie (développement, déploiement, mise à l'échelle).
- **Database per Service** : Chaque microservice possède sa propre base de données privée pour garantir l'isolation des données et l'indépendance technologique.
- **Communication Polyglotte** : Utilisation conjointe de communication **Synchrone** (REST/HTTP) pour les requêtes bloquantes (ex : authentification, lecture) et **Asynchrone** (via Kafka) pour l'échange d'événements et les processus à long terme (ex : notifications, mise à jour de géolocalisation).
- **Conteneurisation** : Utilisation de **Docker** pour empaqueter chaque service, assurant un environnement d'exécution reproductible.
- **Orchestration** : Utilisation de **Kubernetes** pour gérer le déploiement, la découverte de services (*Service Discovery*) et l'auto-scaling.

### 3.2.2 Diagramme d'architecture du système

Le diagramme de la figure 3.1 représente l'architecture technique complète, incluant les microservices, l'API Gateway, le broker de messages Kafka, et l'infrastructure d'exécution (Docker).

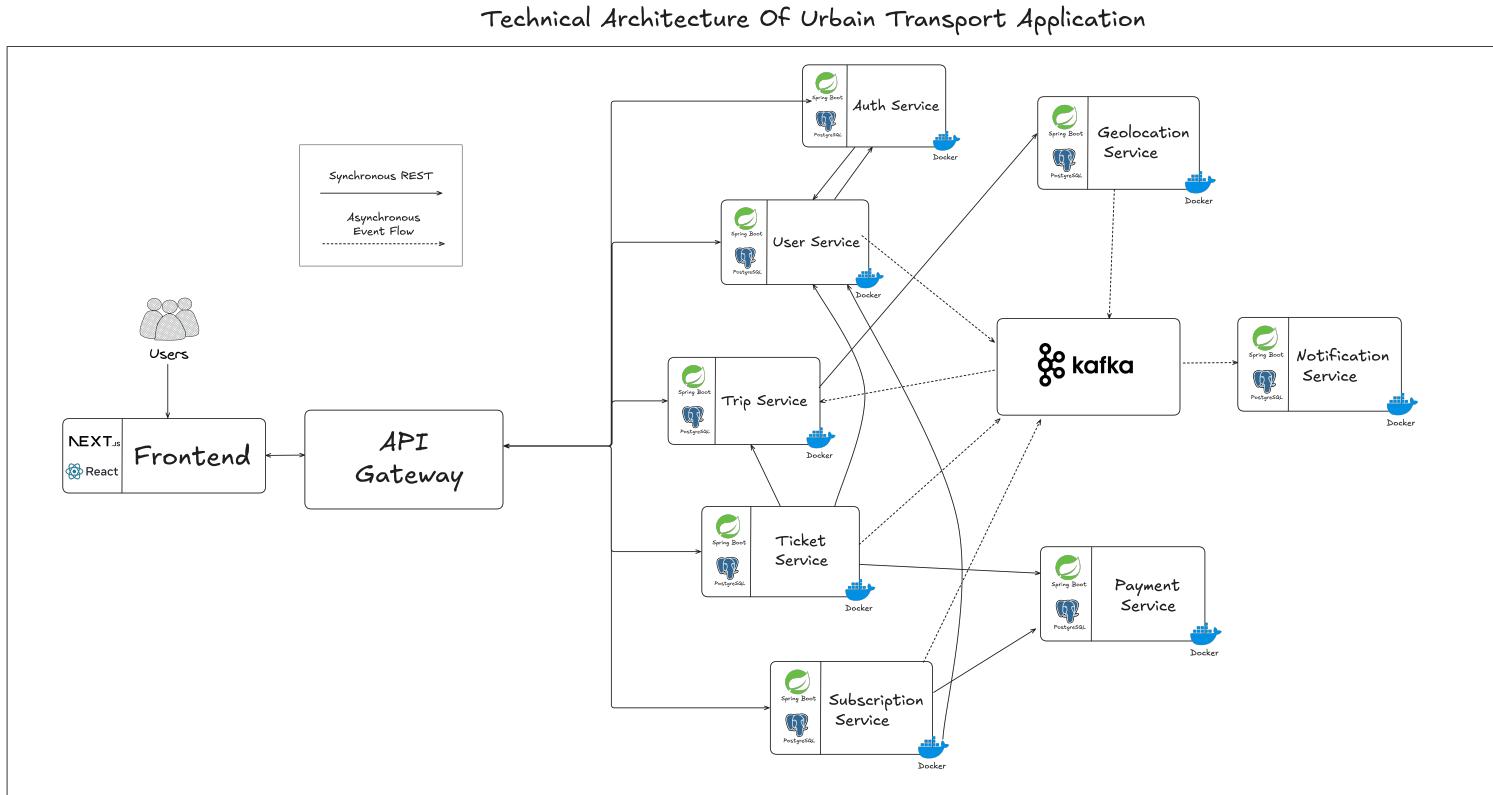


FIGURE 3.1 – Architecture globale du système de Transport Urbain

#### 3.2.2.1 Flux de communication

Le système supporte deux types de flux de communication :

- Flux Synchrone (Requêtes directes)** : Représenté par des flèches pleines. Les **Users** (via le Frontend **NEXT/React**) accèdent au système exclusivement par l'**API Gateway**, qui route la requête vers le service approprié (**Auth Service**, **User Service**, **Trip Service**, etc.) via HTTP/REST. Ces appels sont généralement bloquants et attendent une réponse immédiate.
- Flux Asynchrone (Événements)** : Représenté par des flèches en pointillés. Ces flux sont gérés par le broker de messages **Kafka**. Ils sont utilisés pour propager les changements d'état (événements) entre les services sans bloquer l'appel initial, améliorant ainsi la résilience et le découplage.

## 3.3 Description des services

L'architecture est composée de neuf microservices métier et d'un composant d'infrastructure central (Kafka).

### 3.3.1 Auth Service

- **Rôle** : Gère l'authentification des utilisateurs (connexion, déconnexion). Il est responsable de l'émission et de la validation des **JWT (JSON Web Tokens)** qui sont utilisés par l'**API Gateway** et les autres services pour l'autorisation.
- **Communication** : Principalement synchrone avec l'**API Gateway** et potentiellement avec le **User Service** pour récupérer les rôles de base.
- **Technologie** : Spring Boot (**SpringBoot**).

### 3.3.2 User Service

- **Rôle** : Gère toutes les informations relatives au profil des utilisateurs (passagers, conducteurs, administrateurs).
- **Communication** : Synchrone avec l'**API Gateway**. Asynchrone avec **Kafka** pour publier des événements liés aux modifications de profil (ex : nouvelle inscription) que d'autres services pourraient consommer.
- **Technologie** : Spring Boot (**SpringBoot**).

### 3.3.3 Trip Service

- **Rôle** : Gère les données statiques et dynamiques des trajets (itinéraires, arrêts, horaires planifiés). C'est le cœur de la planification du transport.
- **Communication** : Synchrone (consultation par l'**API Gateway**, le **Ticket Service**). Il consomme des événements du **Geolocation Service** pour mettre à jour les statuts des trajets en temps réel.
- **Technologie** : Spring Boot (**SpringBoot**).

### 3.3.4 Ticket Service

- **Rôle** : Gère le cycle de vie des tickets (achat, annulation, validation, historique).
- **Communication** : Synchrone avec l'**API Gateway**, **Trip Service** et **Payment Service**. Publie des événements asynchrones sur **Kafka** (ex : **TicketPurchased**) consommés notamment par le **Notification Service**.
- **Technologie** : Spring Boot (**SpringBoot**).

### 3.3.5 Subscription Service

- **Rôle** : Gère les formules d'abonnement des utilisateurs (création, renouvellement, expiration).
- **Communication** : Synchrone avec l'**API Gateway** et **Payment Service** pour les transactions récurrentes. Publie des événements (ex : **SubscriptionExpired**) sur **Kafka**.
- **Technologie** : Spring Boot (**SpringBoot**).

### 3.3.6 Geolocation Service

- **Rôle** : Gère le suivi en temps réel de la position des véhicules. Il reçoit en continu les coordonnées GPS des bus.
- **Communication** : Publie les données de localisation en continu sur **Kafka**. Ces événements sont consommés par le **Trip Service** (pour les estimations d'arrivée) et le **Notification Service** (pour les alertes de retard).
- **Technologie** : Spring Boot (**SpringBoot**).

### 3.3.7 Payment Service

- **Rôle** : Traite les transactions financières (achat de tickets, renouvellement d'abonnements) en s'intégrant avec des passerelles de paiement externes.
- **Communication** : Est appelé de manière synchrone par **Ticket Service** et **Subscription Service**. Publie l'événement de résultat (**PaymentSucceeded** ou **PaymentFailed**) sur Kafka pour mettre à jour l'état transactionnel des services appelants.
- **Technologie** : Spring Boot (**SpringBoot**).

### 3.3.8 Notification Service

- **Rôle** : Envoie des notifications aux utilisateurs (email, SMS) en réaction à des événements métier.
- **Communication** : C'est un service purement événementiel. Il **consomme** des événements de **Kafka** provenant de **Ticket Service**, **Subscription Service**, **Geolocation Service**, et **Payment Service** pour déclencher l'envoi d'alertes.
- **Technologie** : Spring Boot (**SpringBoot**).

### 3.3.9 API Gateway

- **Rôle** : Point d'entrée unique pour toutes les requêtes externes. Il assure le **routage** vers le microservice approprié, l'**authentification** (en validant le JWT via l'**Auth Service**), le **rate limiting** et la **terminaison SSL**.
- **Technologie** : Spring Cloud Gateway ou Zuul.

## 3.4 Communication et intégration

### 3.4.1 Communication inter-services (REST et Kafka)

Nous adoptons une stratégie de communication hybride, optimisée pour le type d'opération :

#### 3.4.1.1 Communication Synchrone (REST/HTTP)

Utilisée pour les appels nécessitant une réponse immédiate et pour les requêtes de données simples.

- **Exemple** : L'**API Gateway** appelle l'**Auth Service** pour valider un JWT. Le **Ticket Service** appelle le **Payment Service** pour initier une transaction.

- **Inconvénients gérés** : Pour pallier les problèmes de défaillance réseau ou de latence, les patterns de résilience (Circuit Breaker, Retry, Timeout) sont implémentés.

### 3.4.1.2 Communication Asynchrone (Apache Kafka)

Kafka est utilisé comme **Event Bus** central pour garantir le découplage et la cohérence éventuelle des données (*Eventual Consistency*).

- **Rôle de Kafka** : Fournir un flux de données persistant, partitionné et tolérant aux pannes.
- **Exemple de Flux** :
  1. Le **Ticket Service** publie un événement : `TicketPurchased` sur le topic `tickets.purchased`.
  2. Le **Notification Service** (Consommateur 1) écoute ce topic et envoie un email de confirmation.
  3. Le **Payment Service** publie le statut `PaymentSucceeded` sur `payments.status`.
  4. Le **Ticket Service** (Consommateur 1) met à jour l'état final du ticket en base de données.

### 3.4.2 Gestion des erreurs et tolérance aux pannes

La résilience du système est assurée par l'implémentation de patterns de tolérance aux pannes :

- **Circuit Breaker** : Isoler les services défaillants pour empêcher l'épuisement des ressources (gestion des pannes en cascade).
- **Saga Pattern (Chorégraphie)** : Utilisé pour maintenir la cohérence transactionnelle sur plusieurs microservices via l'échange d'événements Kafka. Par exemple, l'achat d'un ticket implique des étapes dans **Ticket**, **Payment**, et **Notification Services**.
- **Retry with Exponential Backoff** : Réessayer les requêtes échouées temporairement avec des délais croissants.

## 3.5 Sécurité et authentification

### 3.5.1 Mécanismes d'authentification et d'autorisation

- **Authentification (JWT/OAuth2)** : L'**Auth Service** authentifie l'utilisateur (via identifiant/mot de passe), puis émet un **JWT**. Ce token non-expirable est utilisé pour sécuriser chaque requête ultérieure.
- **Validation JWT** : L'**API Gateway** intercepte toutes les requêtes, envoie le **JWT** au **Auth Service** pour validation et extraction des informations de l'utilisateur (ID, Rôles), puis l'injecte dans l'en-tête pour le service final.
- **Autorisation (RBAC)** : Les microservices appliquent le **Role-Based Access Control (RBAC)** en fonction du rôle extrait du **JWT** pour déterminer si l'utilisateur a le droit d'exécuter l'opération demandée.

### 3.5.2 Gestion des secrets

Les identifiants de base de données, les clés d'API et les certificats sont gérés par un gestionnaire de secrets centralisé (HashiCorp Vault ou équivalent Kubernetes Secrets Manager) pour éviter de les exposer dans le code ou les configurations.

## 3.6 Infrastructure et déploiement

### 3.6.1 Conteneurisation et orchestration (Docker, Kubernetes)

- **Docker** : Chaque microservice est conteneurisé. L'image Docker contient l'application et toutes ses dépendances, garantissant une exécution homogène.
- **Kubernetes (K8s)** : Plateforme d'orchestration pour gérer le cycle de vie des conteneurs. K8s assure :
  - La **Scalabilité horizontale** : Les déploiements sont configurés avec des HPA (*Horizontal Pod Autoscaler*) basés sur l'utilisation CPU.
  - La **Découverte de Service** : Les services se trouvent mutuellement via le DNS interne de Kubernetes.
  - L'**Auto-réparation** : K8s redémarre automatiquement les conteneurs défaillants.

### 3.6.2 Base de données décentralisée (Database per Service)

Chaque microservice est associé à sa propre base de données relationnelle (PostgreSQL) conteneurisée.

- **Avantage** : Indépendance technologique, isolation des pannes de base de données et capacité à choisir la base de données optimale pour le besoin du service.
- **Défis** : La gestion des transactions distribuées est résolue par le pattern **Saga** via **Kafka** (cohérence éventuelle).

### 3.6.3 Monitoring et logs (Observabilité)

L'observabilité est cruciale dans un environnement distribué :

- **Logs** : Les logs structurés sont collectés par un système centralisé (**ELK Stack** - Elasticsearch, Logstash, Kibana, ou équivalent Cloud) pour l'analyse des erreurs et la traçabilité.
- **Métriques** : **Prometheus** collecte les métriques de performance (latence, taux d'erreur, CPU) de chaque microservice.
- **Visualisation** : **Grafana** fournit des tableaux de bord en temps réel et des systèmes d'alerte pour les équipes opérationnelles.
- **Tracing Distribué** : Un système comme *Zipkin* ou *Jaeger* est utilisé pour suivre le chemin d'une requête utilisateur à travers tous les microservices qu'elle traverse.

## 3.7 Conclusion

L'architecture microservices adoptée, centrée autour de l'**API Gateway** et du bus d'événements **Kafka**, permet de répondre aux exigences de performance, de résilience et de scalabilité définies dans l'analyse fonctionnelle. Le choix des technologies Cloud Native (Docker, Kubernetes) assure un déploiement et une gestion en production modernes et efficaces, posant ainsi les bases d'un système robuste et évolutif pour l'entreprise de transport urbain.

# Chapitre 4

## Conception détaillée

### 4.1 Introduction

Ce chapitre présente la modélisation technique détaillée de chaque microservice du système de transport urbain. Il traduit les exigences fonctionnelles en spécifications techniques concrètes, avec pour chaque service : les diagrammes de classes, les schémas de base de données, les API REST et les flux d’interaction. Cette modélisation précise sert de blueprint pour le développement et assure la cohérence de l’architecture microservices.

### 4.2 Conception par service

#### 4.2.1 Service Authentification (Auth Service)

##### 4.2.1.1 Objectif

Gérer l’authentification des utilisateurs (passagers, conducteurs, administrateurs), émettre et valider les JWT tokens, gérer la réinitialisation des mots de passe et la validation des emails.

##### 4.2.1.2 Cas d’utilisation

###### Principaux cas d’utilisation :

- Connexion d’un utilisateur (login)
- Déconnexion d’un utilisateur (logout)
- Réinitialisation du mot de passe
- Vérification de l’email
- Validation JWT pour sécuriser les endpoints

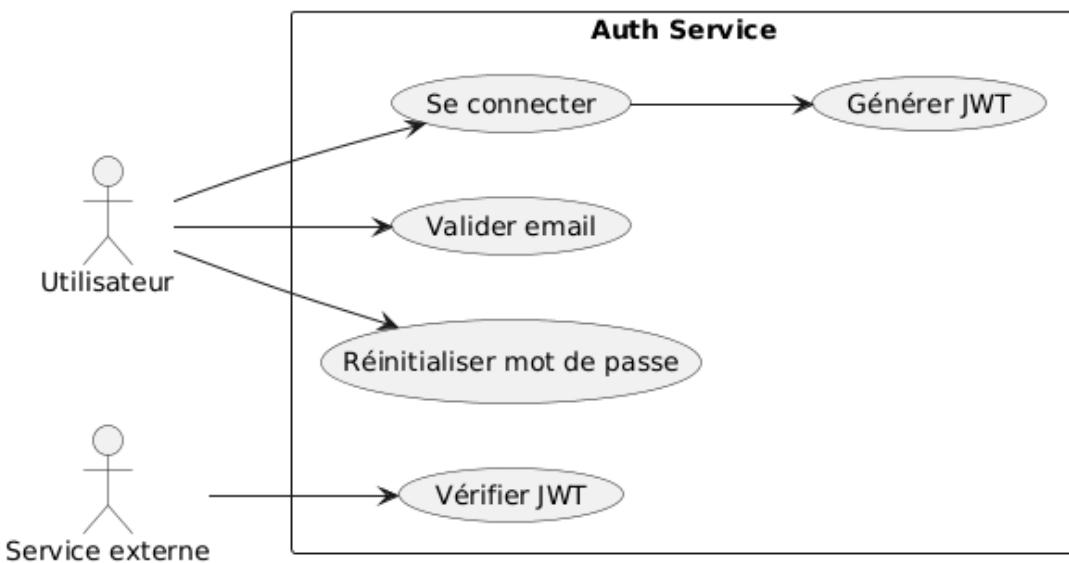


FIGURE 4.1 – Diagramme de cas d'utilisation du service Authentification

#### 4.2.1.3 Diagramme de classes

Classes principales pour Auth Service :

- Utilisateur : informations de base, email, rôle
- Token : JWT ou refresh token
- TentativeConnexion : suivi des tentatives de connexion

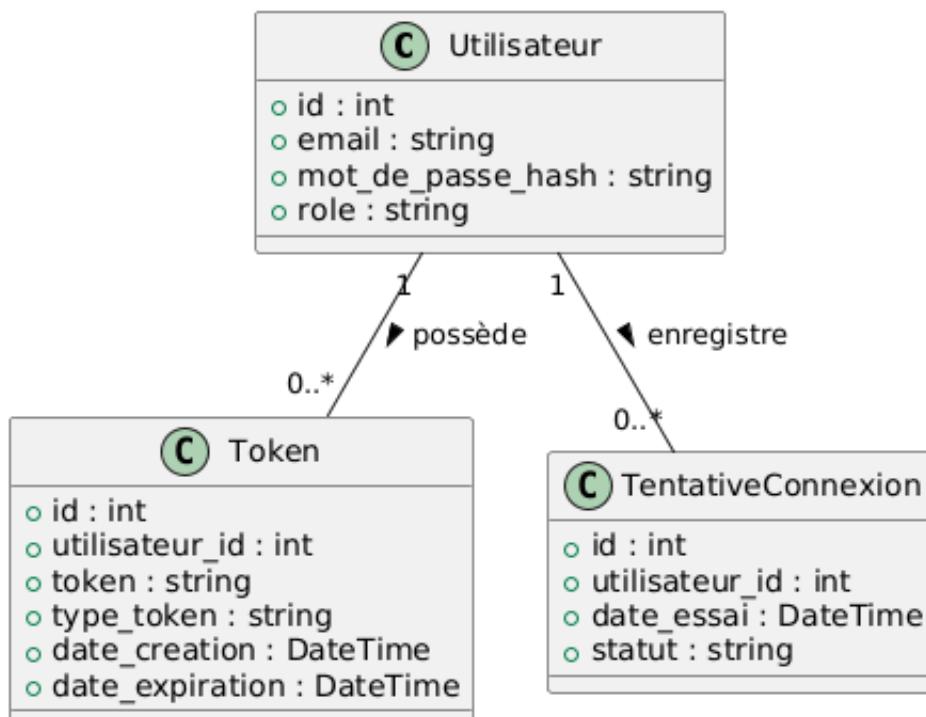


FIGURE 4.2 – Diagramme de classes du service Authentification

#### 4.2.1.4 Base de données et entités (Database per Service)

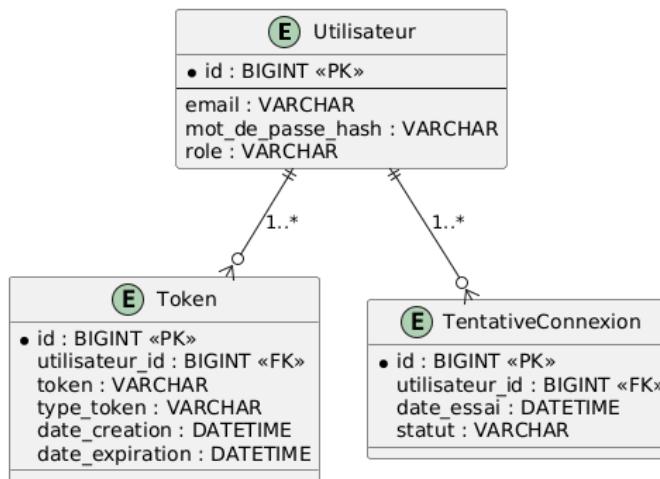


FIGURE 4.3 – Architecture de la base de données pour le service Authentification

Table	Champs
Utilisateur	id (PK), email, mot_de_passe_hash, role
Token	id (PK), utilisateur_id (FK), token, type_token (access/refresh), date_creation, date_expiration
TentativeConnexion	id (PK), utilisateur_id (FK), date_essai, statut

#### 4.2.1.5 API REST / Endpoints

Endpoint	Méthode	Description
/auth/login	POST	Connexion utilisateur et génération JWT
/auth/logout	POST	Déconnexion utilisateur et invalidation du token
/auth/password-reset-request	POST	Demande réinitialisation mot de passe
/auth/password-reset	POST	Réinitialisation du mot de passe avec token
/auth/verify-email	GET	Vérification de l'email via token
/auth/validate-jwt	POST	Vérification JWT pour sécuriser les endpoints

#### 4.2.1.6 Diagramme de séquence – Connexion utilisateur

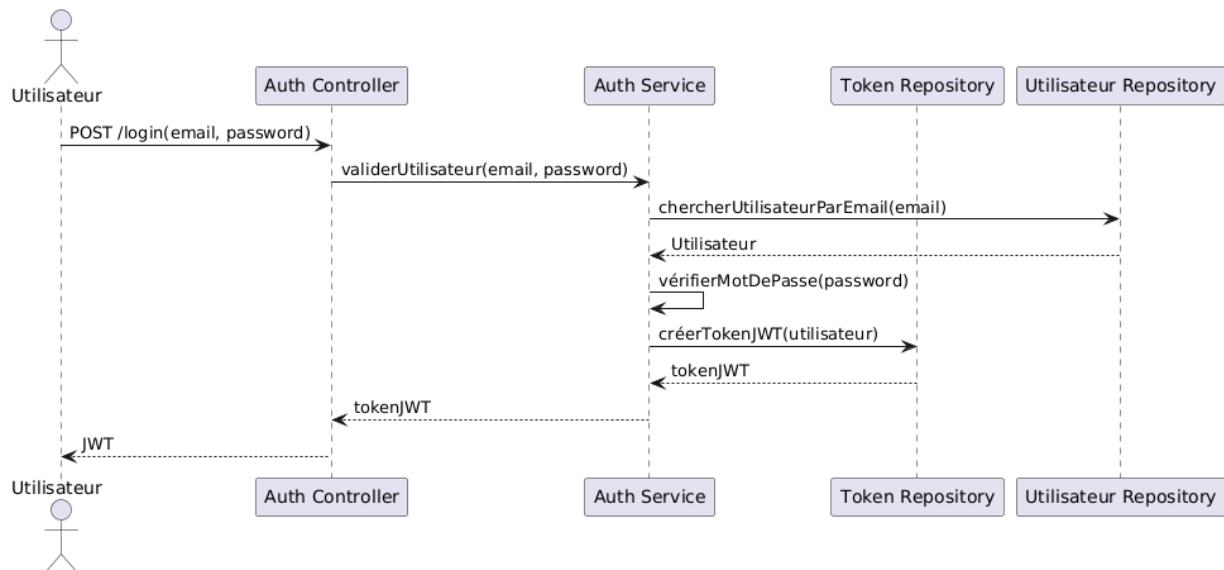


FIGURE 4.4 – Diagramme de séquence du processus de connexion utilisateur

#### 4.2.2 Service Gestion des Utilisateurs (User Service)

##### 4.2.2.1 Objectif

Gérer les informations des utilisateurs : inscription, modification, suppression et fournir les données nécessaires aux autres services (rôle, profil).

##### 4.2.2.2 Cas d'utilisation

**Principaux cas d'utilisation :**

- S'inscrire
- Se connecter
- Modifier son profil
- Supprimer son compte
- Consulter son profil
- Fournir rôle / informations aux autres services

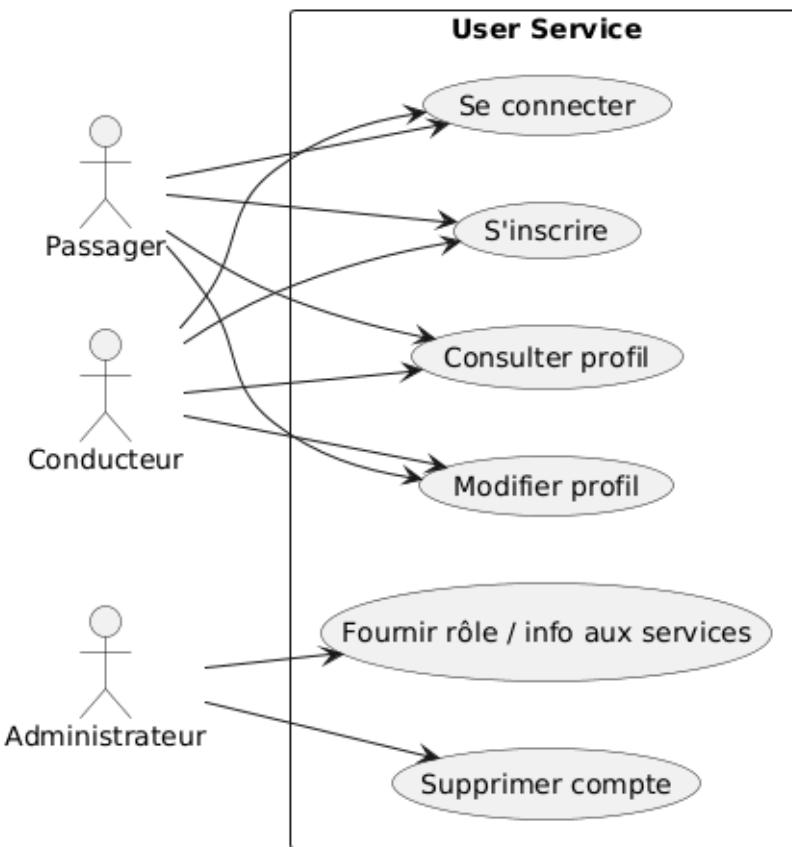


FIGURE 4.5 – Diagramme de cas d'utilisation du service Gestion des Utilisateurs

#### 4.2.2.3 Diagramme de classes

Classes principales pour User Service :

- Utilisateur : informations générales (nom, prénom, email, rôle, téléphone)
- Conducteur : informations spécifiques au conducteur (licence, disponibilités)
- Passager : informations spécifiques au passager (abonnement actif, points fidélité)

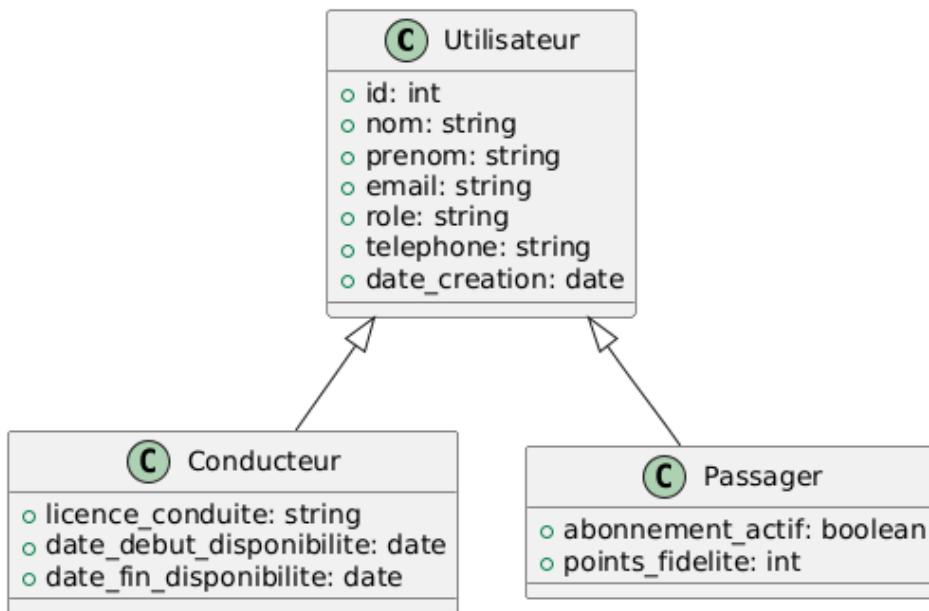


FIGURE 4.6 – Diagramme de classes du service Gestion des Utilisateurs

#### 4.2.2.4 Base de données et entités (Database per Service)

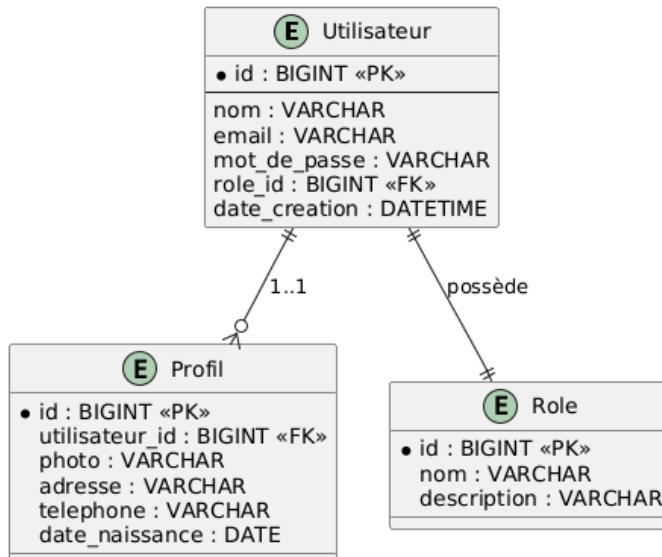


FIGURE 4.7 – Architecture de la base de données pour le service Gestion des Utilisateurs

Table	Champs
Utilisateur	id (PK), nom, prenom, email, role, telephone, date_creation
Conducteur	id (PK, FK Utilisateur), licence_conduite, date_debut_disponibilite, date_fin_disponibilite
Passager	id (PK, FK Utilisateur), abonnement_actif, points_fidelite

#### 4.2.2.5 API REST / Endpoints

Endpoint	Méthode	Description
/user/register	POST	Inscription d'un nouvel utilisateur
/user/login	POST	Connexion utilisateur (vérification via Auth Service)
/user/update/id	PUT	Modification des informations d'un utilisateur
/user/delete/id	DELETE	Suppression d'un utilisateur
/user/id	GET	Consultation des informations d'un utilisateur
/user/role/id	GET	Fournir rôle et permissions aux autres services

#### 4.2.2.6 Diagramme de séquence – Inscription utilisateur

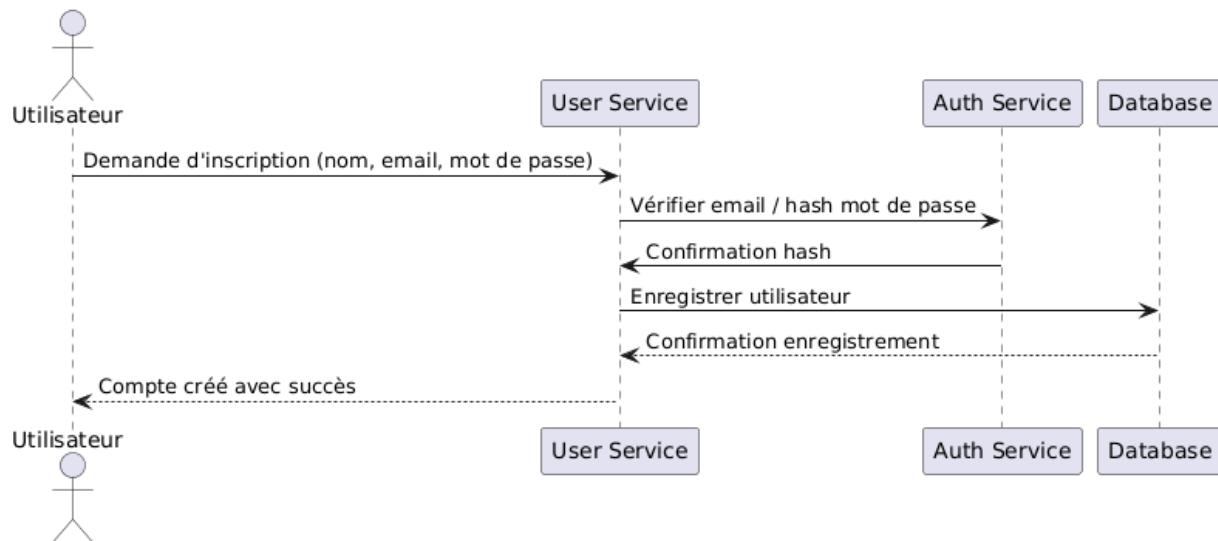


FIGURE 4.8 – Diagramme de séquence du processus d'inscription utilisateur

#### 4.2.3 Service Achat de Tickets (Ticket Service)

##### 4.2.3.1 Objectif

Permettre aux utilisateurs d'acheter, annuler ou consulter leurs tickets. Ce service gère également le suivi des paiements et l'historique des achats. Il interagit principalement avec les services **User Service**, **Trip Service**, **Payment Service** et **Notification Service**.

##### 4.2.3.2 Cas d'utilisation

**Principaux cas d'utilisation :**

- Achat d'un ticket pour un trajet donné
- Annulation d'un ticket avant le départ
- Consultation de l'historique des achats
- Affichage du ticket (QR code, référence)
- Gestion des tarifs et disponibilités (par administrateur ou conducteur)

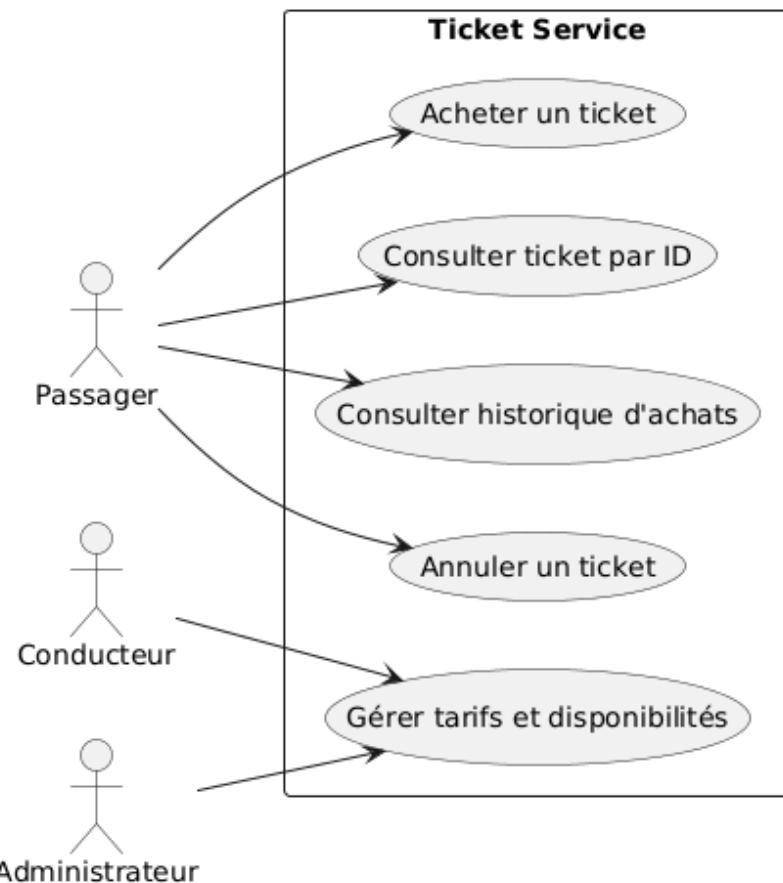


FIGURE 4.9 – Diagramme de cas d'utilisation du service d'achat de tickets

#### 4.2.3.3 Diagramme de classes

Classes principales pour le Ticket Service :

- **Ticket** : représente un ticket acheté par un passager pour un trajet donné.
- **Trajet** : informations sur le trajet associé au ticket.
- **Paiement** : informations relatives au paiement du ticket.

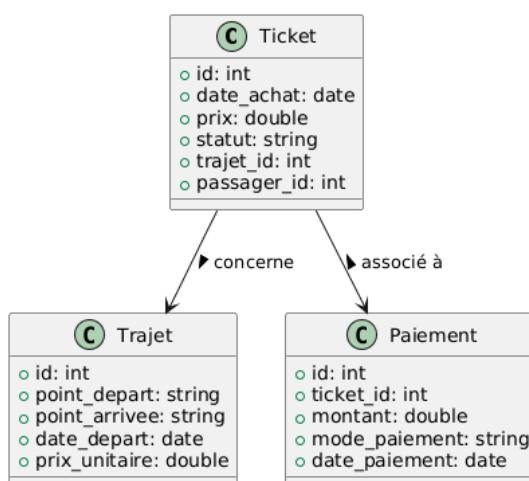


FIGURE 4.10 – Diagramme de classes du service d'achat de tickets

#### 4.2.3.4 Base de données et entités (Database per Service)

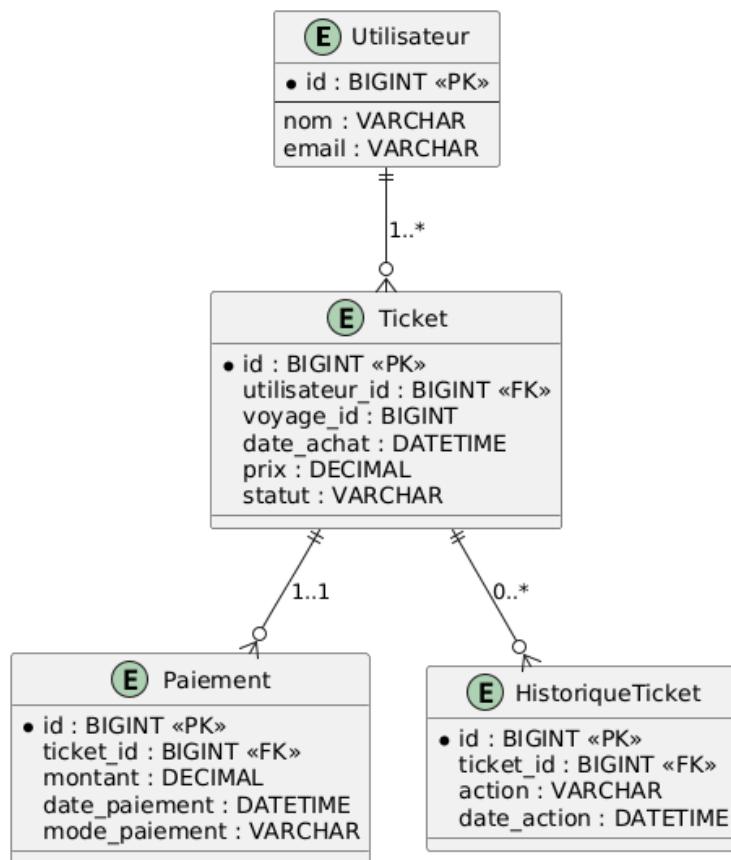


FIGURE 4.11 – Schéma de la base de données pour le service d'achat de tickets

Chaque service dispose de sa propre base de données. Le Ticket Service comprend deux tables principales : `ticket` et `paiement_ticket`.

Table	Champs
Ticket	id (PK), passager_id (FK), trajet_id (FK), date_achat, statut
Paiement_Ticket	id (PK), ticket_id (FK), montant, date_paiement, mode_paiement

#### 4.2.3.5 API REST / Endpoints

Endpoints principaux du Ticket Service :

Endpoint	Méthode	Description
/api/tickets	GET	Récupérer tous les tickets d'un utilisateur
/api/tickets/id	GET	Consulter un ticket spécifique par son identifiant
/api/tickets	POST	Acheter un ticket pour un trajet donné
/api/tickets/id/cancel	PUT	Annuler un ticket avant le départ
/api/tickets/history	GET	Consulter l'historique complet des achats

#### 4.2.3.6 Diagramme de séquence – Achat d'un ticket

Ce diagramme illustre le processus complet d'achat d'un ticket par un passager.

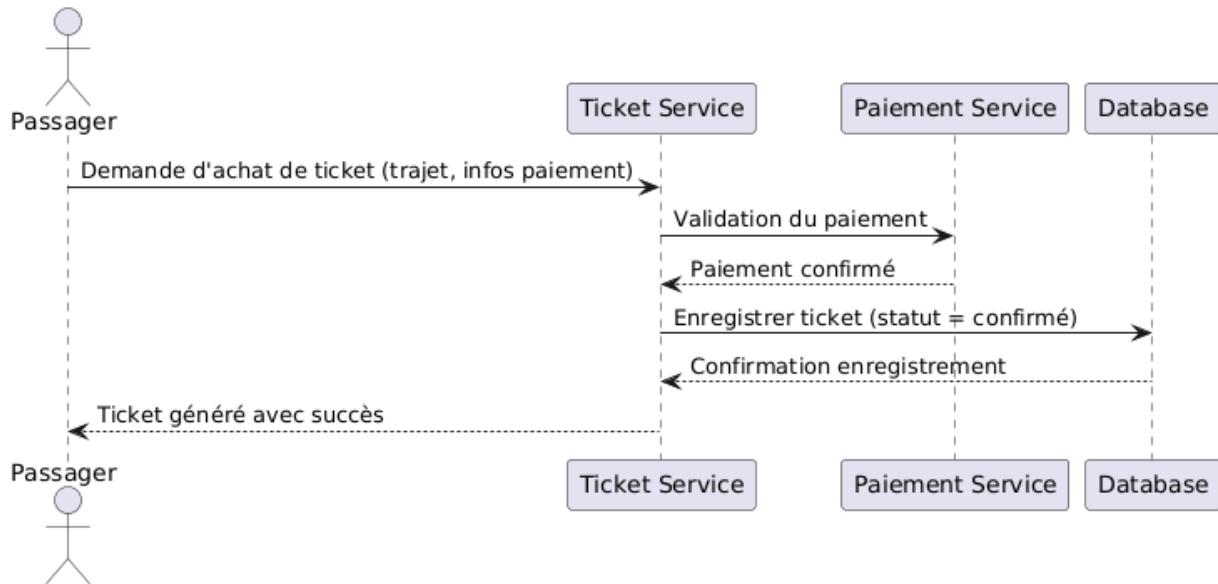


FIGURE 4.12 – Diagramme de séquence du processus d'achat de ticket

#### 4.2.3.7 Interactions avec les autres services

- **Auth Service** : Vérification JWT pour sécuriser les requêtes REST.
- **User Service** : Validation de l'identité du passager.
- **Trip Service** : Récupération des informations sur le trajet choisi.
- **Payment Service** : Validation et traitement du paiement.
- **Notification Service** : Envoi de notifications de confirmation par email/SMS via Kafka.

#### 4.2.3.8 Résumé du flux global

Lorsqu'un utilisateur achète un ticket :

1. Il envoie une requête POST `/api/tickets` avec les détails du trajet et du paiement.
2. Le service valide le JWT via Auth Service.
3. Le paiement est vérifié via Payment Service.
4. Le ticket est enregistré en base avec le statut "confirmé".
5. Une notification est envoyée au passager via Notification Service.

##### Résumé du Service

Le **Ticket Service** est un élément central de l'application de transport urbain. Il connecte les modules *Utilisateurs*, *Trajets*, *Paiements* et *Notifications*, tout en maintenant un historique complet des transactions.

#### 4.2.4 Service Gestion des Abonnements (Subscription Service)

##### 4.2.4.1 Objectif

Gérer le cycle de vie des abonnements des utilisateurs : création, renouvellement, annulation et expiration automatique. Ce service permet de centraliser la gestion des abonnements pour les passagers ou les conducteurs selon leurs besoins, tout en assurant la communication avec le service de paiement pour valider les transactions liées aux abonnements.

##### 4.2.4.2 Cas d'utilisation

**Principaux cas d'utilisation :**

- Création d'un nouvel abonnement
- Renouvellement automatique ou manuel d'un abonnement
- Annulation d'un abonnement actif
- Consultation de l'historique des abonnements
- Détection et désactivation des abonnements expirés

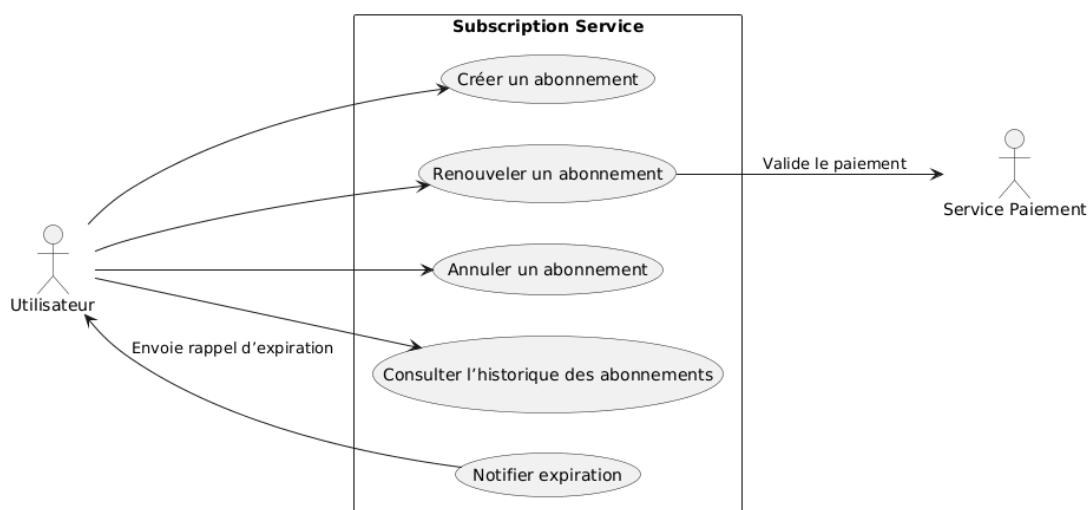


FIGURE 4.13 – Diagramme de cas d'utilisation du service Gestion des Abonnements

##### 4.2.4.3 Diagramme de classes

**Classes principales pour le Subscription Service :**

- **Utilisateur** : représente le client ou conducteur lié à un abonnement.
- **Abonnement** : contient les détails de l'abonnement (type, statut, dates).
- **Paiement** : enregistre les transactions liées à l'abonnement.
- **Notification** : permet d'envoyer des rappels de renouvellement ou d'expiration.

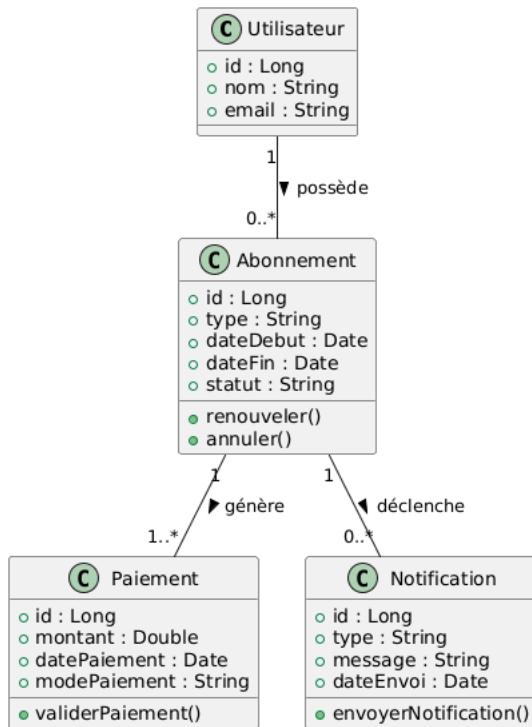


FIGURE 4.14 – Diagramme de classes du service Gestion des Abonnements

#### 4.2.4.4 Base de données et entités (Database per Service)

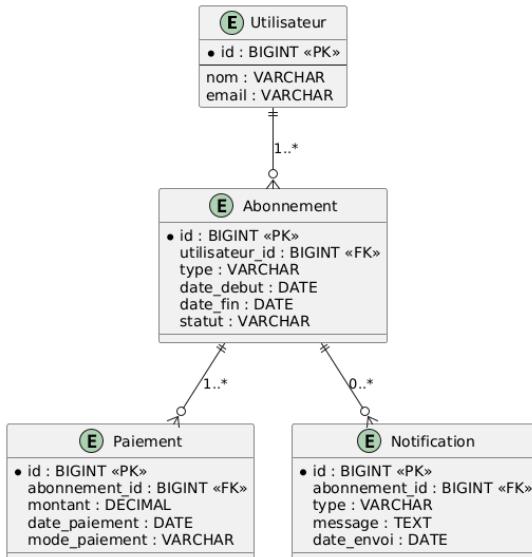


FIGURE 4.15 – Architecture de la base de données pour le service Gestion des Abonnements

Table	Champs
Utilisateur	id (PK), nom, email
Abonnement	id (PK), utilisateur_id (FK), type, date_debut, date_fin, statut (actif/expiré/annulé)

Paiement	id (PK), abonnement_id (FK), montant, date_paiement, mode_paiement
Notification	id (PK), abonnement_id (FK), type, message, date_envoi

#### 4.2.4.5 API REST / Endpoints

Endpoint	Méthode	Description
/subscriptions	POST	Créer un nouvel abonnement pour un utilisateur
/subscriptions/{id}/renew	POST	Renouveler un abonnement existant
/subscriptions/{id}/cancel	PUT	Annuler un abonnement actif
/subscriptions/{userId}	GET	Récupérer les abonnements d'un utilisateur
/subscriptions/expired	GET	Lister les abonnements expirés

#### 4.2.4.6 Diagramme de séquence – Renouvellement d'un abonnement

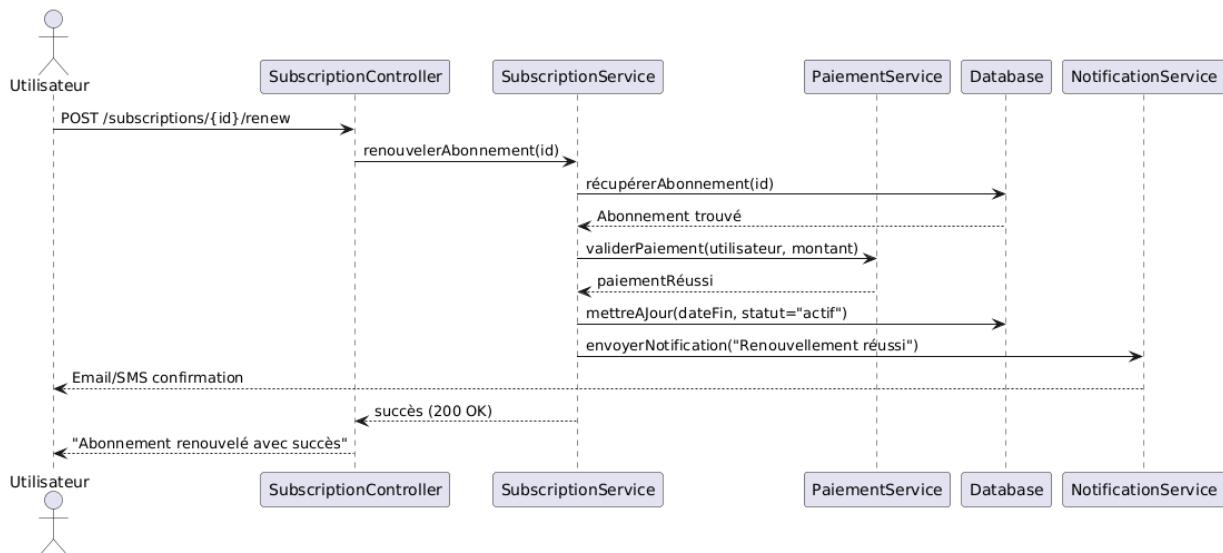


FIGURE 4.16 – Diagramme de séquence du processus de renouvellement d'un abonnement

#### 4.2.5 Service Gestion des Trajets et Horaires (Trip Service)

##### 4.2.5.1 Objectif

Gérer l'ensemble des informations relatives aux routes (lignes de bus), trajets, horaires et arrêts. Ce service permet aux passagers de consulter les horaires disponibles, rechercher des trajets par origine/destination, et aux conducteurs de consulter leurs trajets assignés. Les administrateurs peuvent créer, modifier et gérer les routes, horaires et assignations.

##### 4.2.5.2 Cas d'utilisation

###### Principaux cas d'utilisation :

- Consulter les routes disponibles

- Consulter les horaires
- Rechercher un trajet (origine/destination)
- Consulter les arrêts d'une route
- Consulter mes trajets assignés (Conducteur)
- Créer/Modifier une route (Admin)
- Créer/Modifier un horaire (Admin)
- Gérer les arrêts (Admin)
- Assigner un conducteur à un trajet (Admin)
- Annuler/Modifier un trajet (Admin)

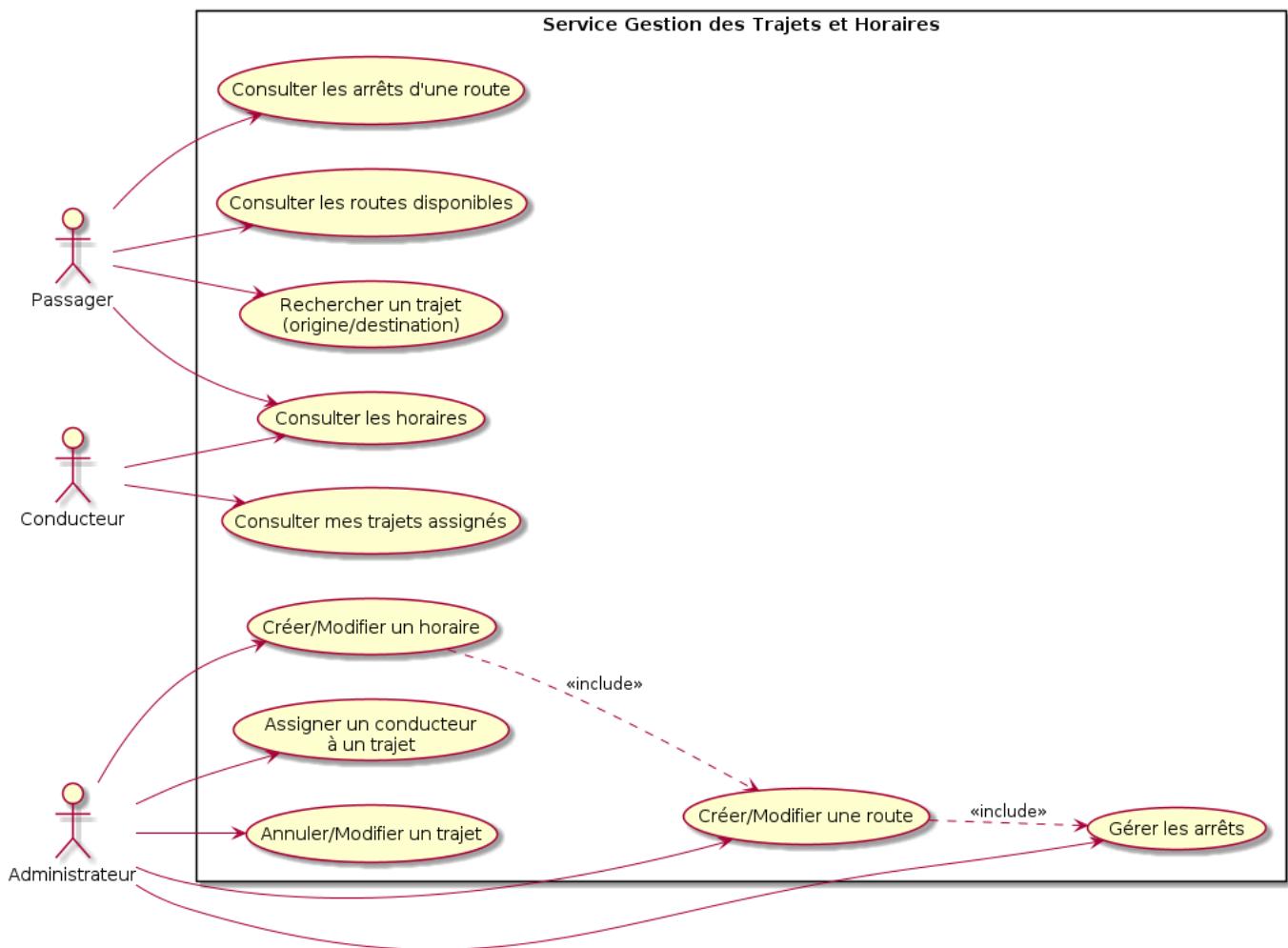


FIGURE 4.17 – Diagramme de cas d'utilisation du service Gestion des Trajets et Horaires

#### 4.2.5.3 Diagramme de classes

##### Classes principales pour Trip Service :

- Route : représente une ligne de bus (nom, numéro, distance, durée estimée)

- **Stop** : représente un arrêt de bus (nom, adresse, coordonnées GPS)
- **RouteStop** : association entre une route et ses arrêts avec ordre et temps d'arrêt
- **Schedule** : horaires planifiés pour une route (heure départ/arrivée, jours, fréquence)
- **Trip** : instance d'un trajet à une date donnée avec conducteur et bus assignés
- **TripStop** : suivi des arrêts réels d'un trajet (heures prévues et réelles)

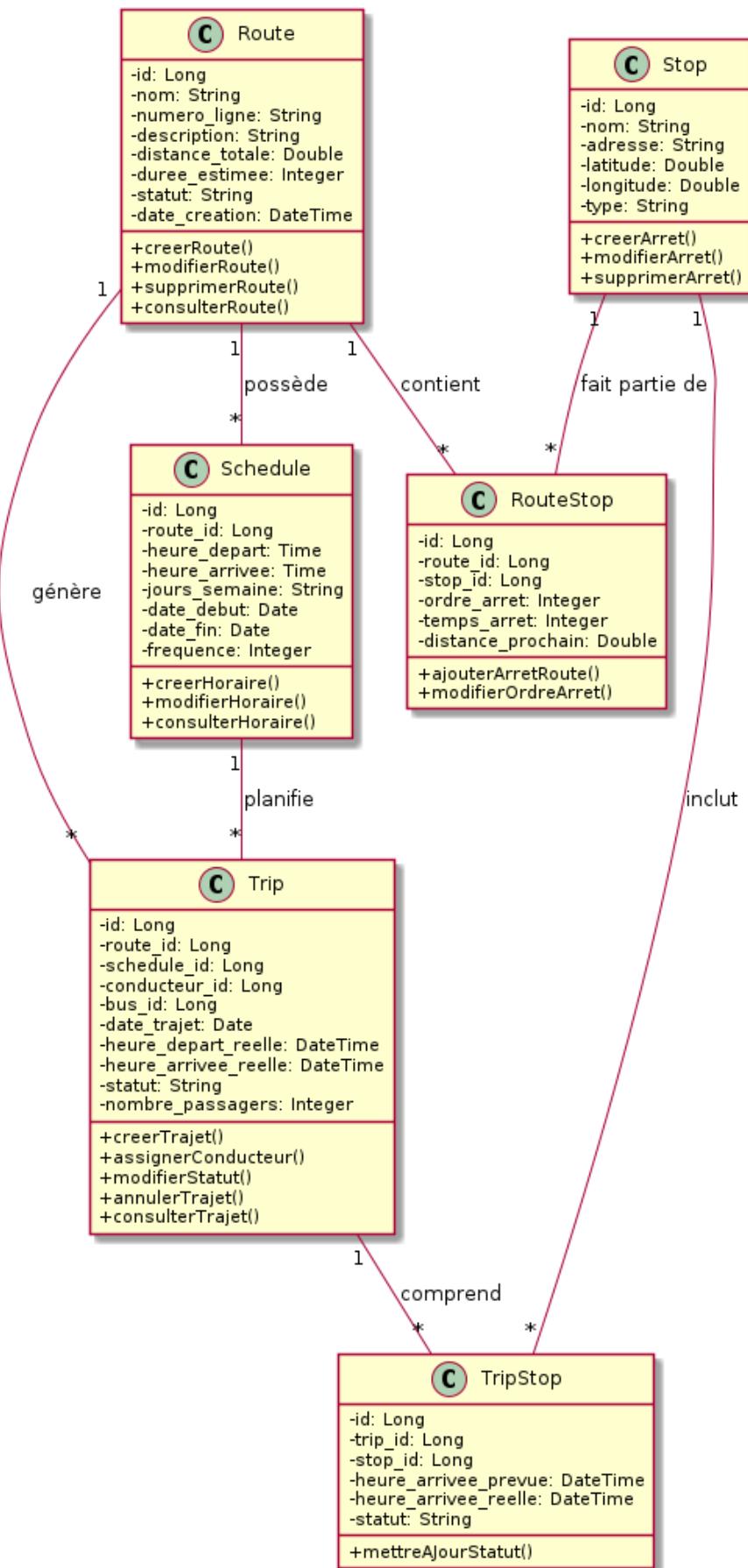


FIGURE 4.18 – Diagramme de classes du service Gestion des Trajets et Horaires

#### 4.2.5.4 Base de données et entités (Database per Service)

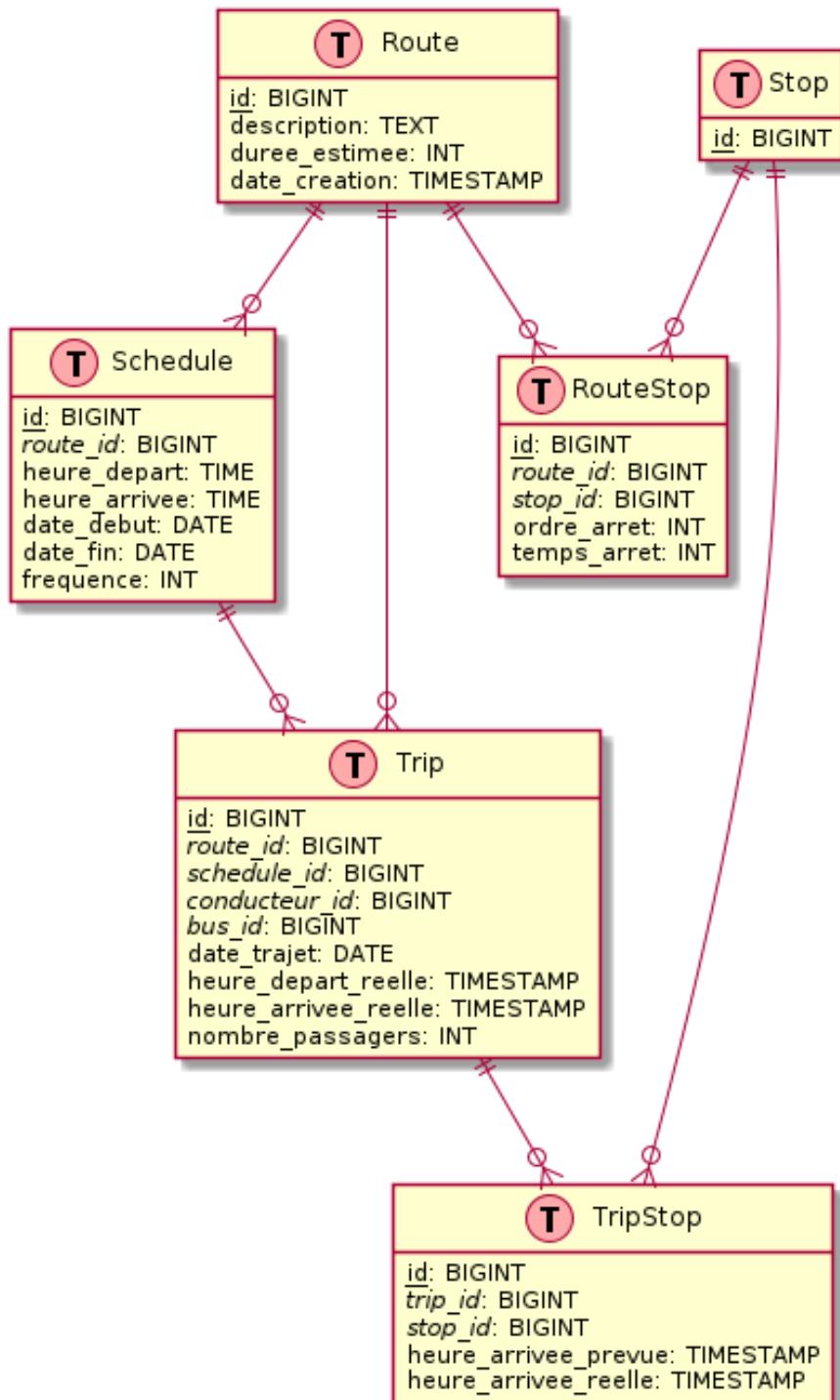


FIGURE 4.19 – Architecture de la base de données pour le service Gestion des Trajets

Table	Champs
Route	id (PK), nom, numero_ligne, description, distance_totale, duree_estimee, statut, date_creation
Stop	id (PK), nom, adresse, latitude, longitude, type

RouteStop	id (PK), route_id (FK), stop_id (FK), ordre_arret, temps_arret, distance_prochain
Schedule	id (PK), route_id (FK), heure_depart, heure_arrivee, jours_semaine, date_debut, date_fin, frequence
Trip	id (PK), route_id (FK), schedule_id (FK), conducteur_id (FK), bus_id (FK), date_trajet, heure_depart_reelle, heure_arrivee_reelle, statut, nombre_passagers
TripStop	id (PK), trip_id (FK), stop_id (FK), heure_arrivee_prevue, heure_arrivee_reelle, statut

#### 4.2.5.5 API REST / Endpoints

Endpoint	Méthode	Description
/routes	GET	Liste toutes les routes disponibles
/routes/{id}	GET	Détails d'une route spécifique
/routes	POST	Créer une nouvelle route (Admin)
/routes/{id}	PUT	Modifier une route (Admin)
/routes/{id}/stops	GET	Liste des arrêts d'une route
/stops	GET	Liste tous les arrêts
/stops	POST	Créer un nouvel arrêt (Admin)
/schedules/route/{routeId}	GET	Horaires d'une route
/schedules	POST	Créer un nouvel horaire (Admin)
/trips/search	GET	Rechercher trajets par origine/destination/date
/trips/{id}	GET	Détails d'un trajet spécifique
/trips/driver/{driverId}	GET	Trajets assignés à un conducteur
/trips	POST	Créer un nouveau trajet (Admin)
/trips/{id}/assign	PUT	Assigner conducteur/bus à un trajet (Admin)
/trips/{id}/status	PUT	Modifier statut d'un trajet
/trips/{id}/cancel	PUT	Annuler un trajet (Admin)

#### 4.2.5.6 Diagramme de séquence – Recherche de trajet par passager

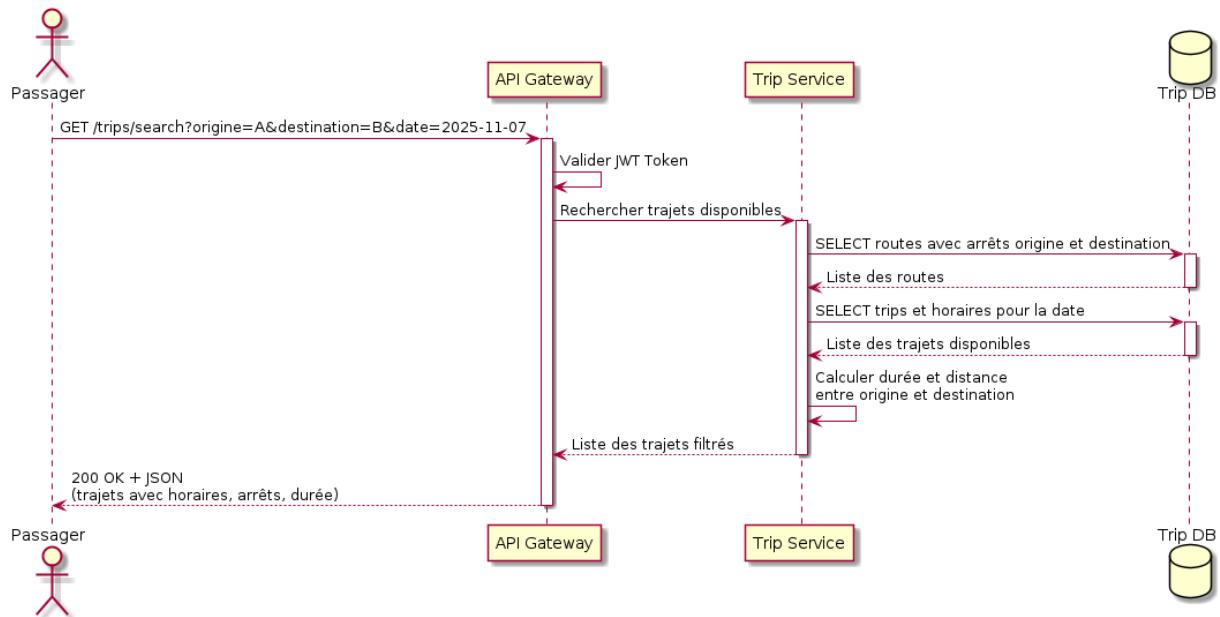


FIGURE 4.20 – Diagramme de séquence du processus de recherche de trajet

#### 4.2.6 Service Géolocalisation des Bus (Geolocation Service)

##### 4.2.6.1 Objectif

Assurer le suivi en temps réel de la position des bus via GPS, calculer les temps d'arrivée estimés (ETA), détecter les déviations de route, et conserver l'historique des positions. Ce service s'intègre avec le Trip Service pour mettre à jour dynamiquement les informations de localisation et alerter en cas d'anomalie.

##### 4.2.6.2 Cas d'utilisation

###### Principaux cas d'utilisation :

- Suivre position d'un bus en temps réel
- Consulter historique de position
- Envoyer position GPS (Conducteur/Système GPS)
- Calculer temps d'arrivée estimé (ETA)
- Visualiser tous les bus sur une carte (Admin)
- Activer/Désactiver tracking (Conducteur)
- Recevoir alertes de déviation (Admin)
- Générer rapports de trajet (Admin)

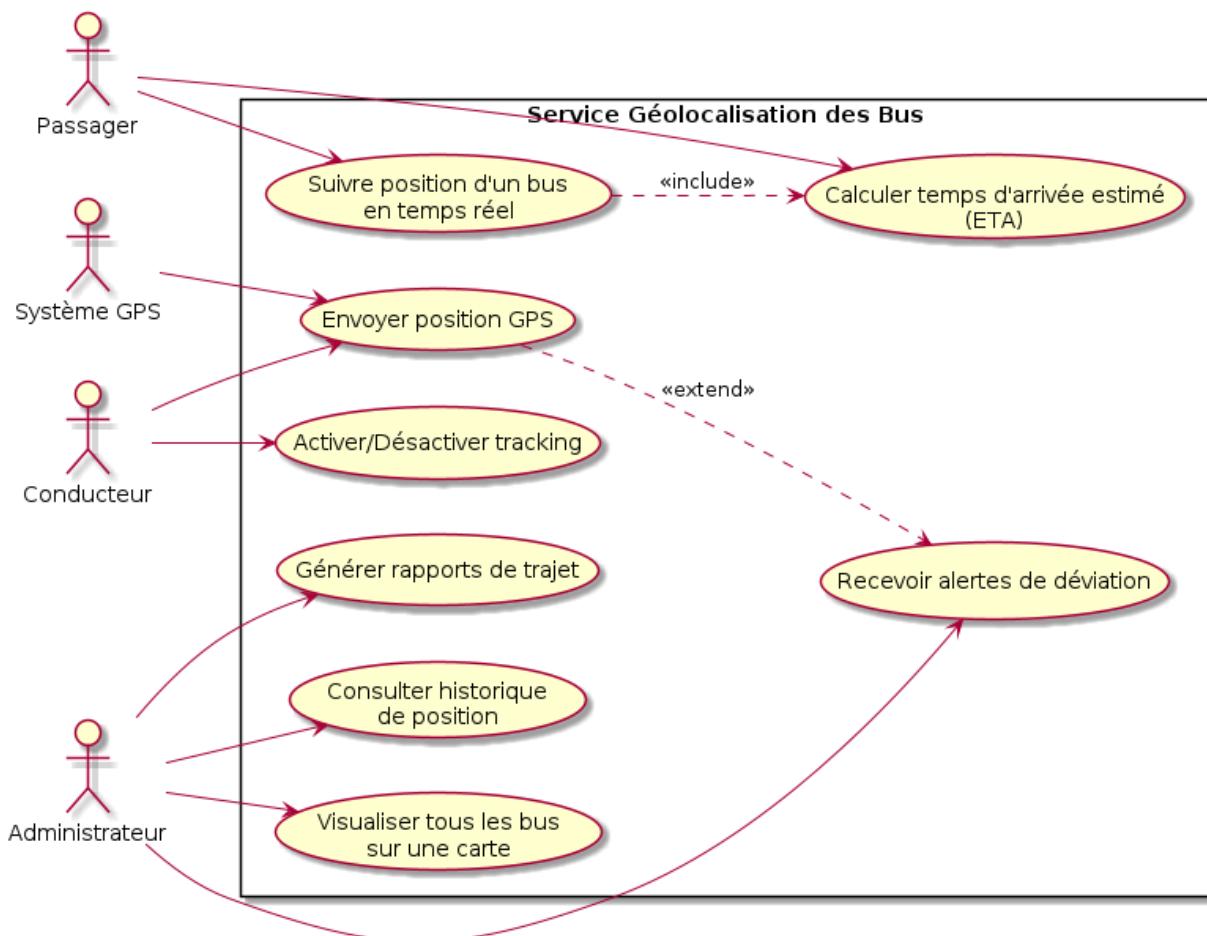
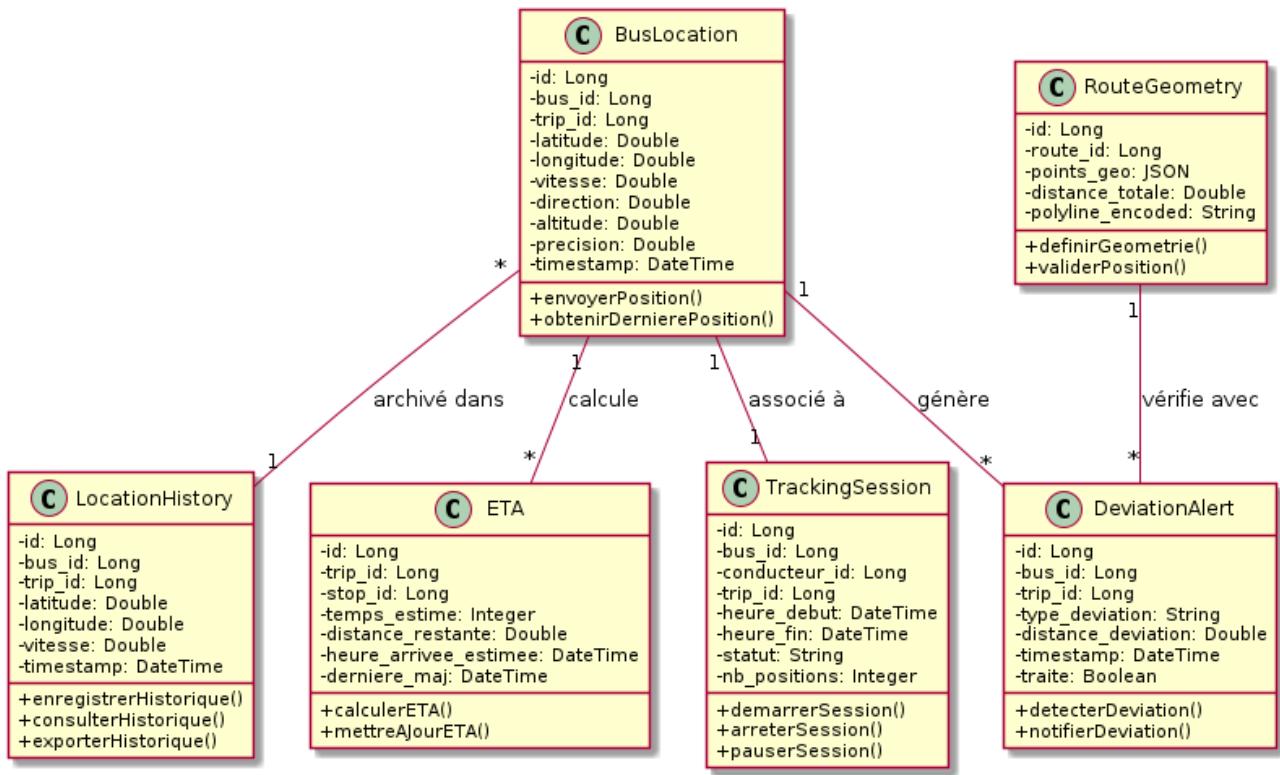


FIGURE 4.21 – Diagramme de cas d'utilisation du service Géolocalisation

#### 4.2.6.3 Diagramme de classes

Classes principales pour Geolocation Service :

- **BusLocation** : position actuelle d'un bus (latitude, longitude, vitesse, direction)
- **LocationHistory** : historique des positions pour analyse et reporting
- **ETA** : temps d'arrivée estimé à chaque arrêt
- **TrackingSession** : session de tracking d'un trajet (début, fin, statut)
- **DeviationAlert** : alertes de déviation par rapport à la route prévue
- **RouteGeometry** : géométrie de la route pour validation des positions



#### 4.2.6.4 Base de données et entités (Database per Service)

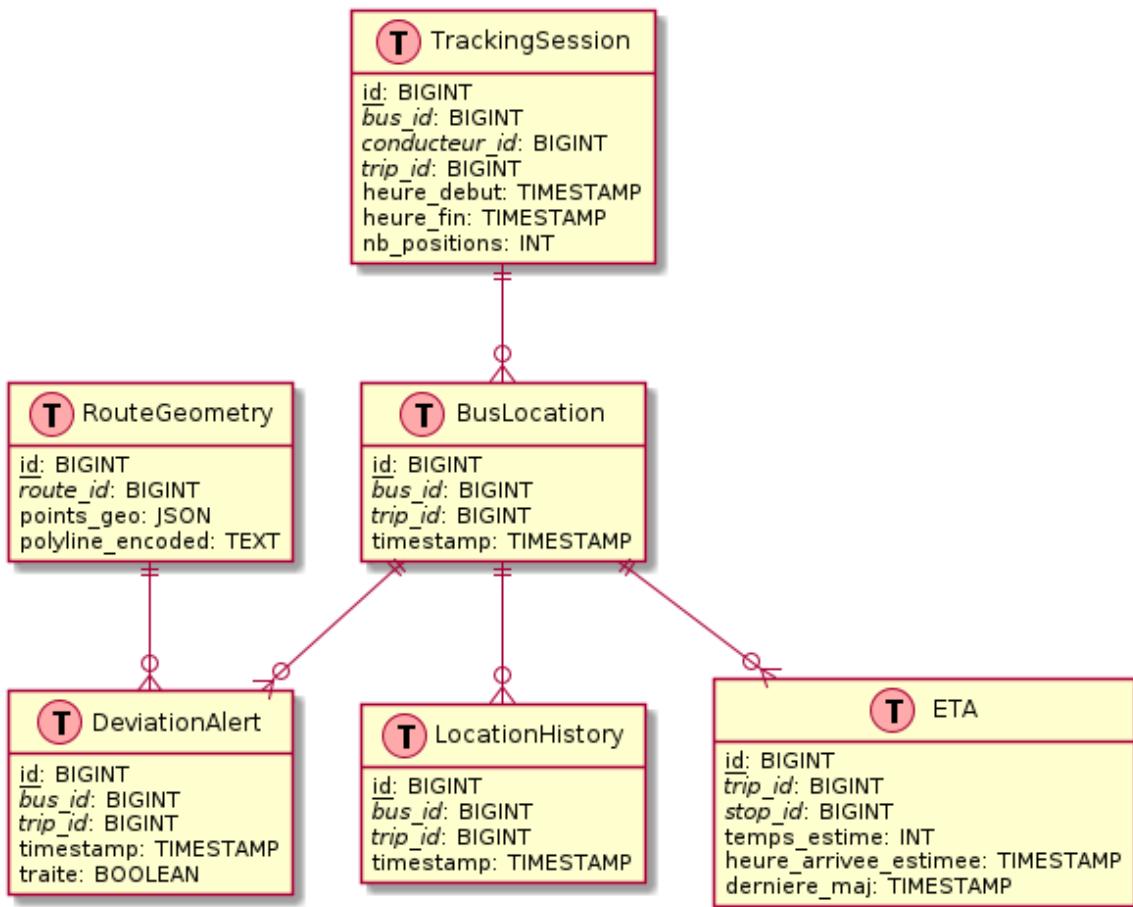


FIGURE 4.23 – Architecture de la base de données pour le service Géolocalisation

Table	Champs
BusLocation	id (PK), bus_id (FK), trip_id (FK), latitude, longitude, vitesse, direction, altitude, precision, timestamp
LocationHistory	id (PK), bus_id (FK), trip_id (FK), latitude, longitude, vitesse, timestamp
ETA	id (PK), trip_id (FK), stop_id (FK), temps_estime, distance_restante, heure_arrivee_estimee, derniere_maj
TrackingSession	id (PK), bus_id (FK), conducteur_id (FK), trip_id (FK), heure_debut, heure_fin, statut, nb_positions
DeviationAlert	id (PK), bus_id (FK), trip_id (FK), type_deviation, distance_deviation, timestamp, traite
RouteGeometry	id (PK), route_id (FK), points_geo (JSON), distance_totale, polyline_encoded

#### 4.2.6.5 API REST / Endpoints

Endpoint	Méthode	Description
/geolocation/session/start	POST	Démarrer une session de tracking
/geolocation/session/{id}/stop	PUT	Arrêter une session de tracking
/geolocation/position	POST	Enregistrer position GPS du bus
/geolocation/bus/{busId}/current	GET	Position actuelle d'un bus
/geolocation/trip/{tripId}/buses	GET	Positions de tous les bus d'un trajet
/geolocation/buses/active	GET	Positions de tous les bus actifs (carte)
/geolocation/history/bus/{busId}	GET	Historique de position d'un bus
/geolocation/eta/trip/{tripId}	GET	ETA pour tous les arrêts d'un trajet
/geolocation/eta/stop/{stopId}	GET	ETA pour tous les bus arrivant à un arrêt
/geolocation/deviations/active	GET	Liste des alertes de déviation actives
/geolocation/deviations/{id}/resolve	PUT	Marquer une déviation comme traitée
/geolocation/route/{routeId}/geometry	GET	Géométrie d'une route

#### 4.2.6.6 Diagramme de séquence – Suivi en temps réel d'un bus

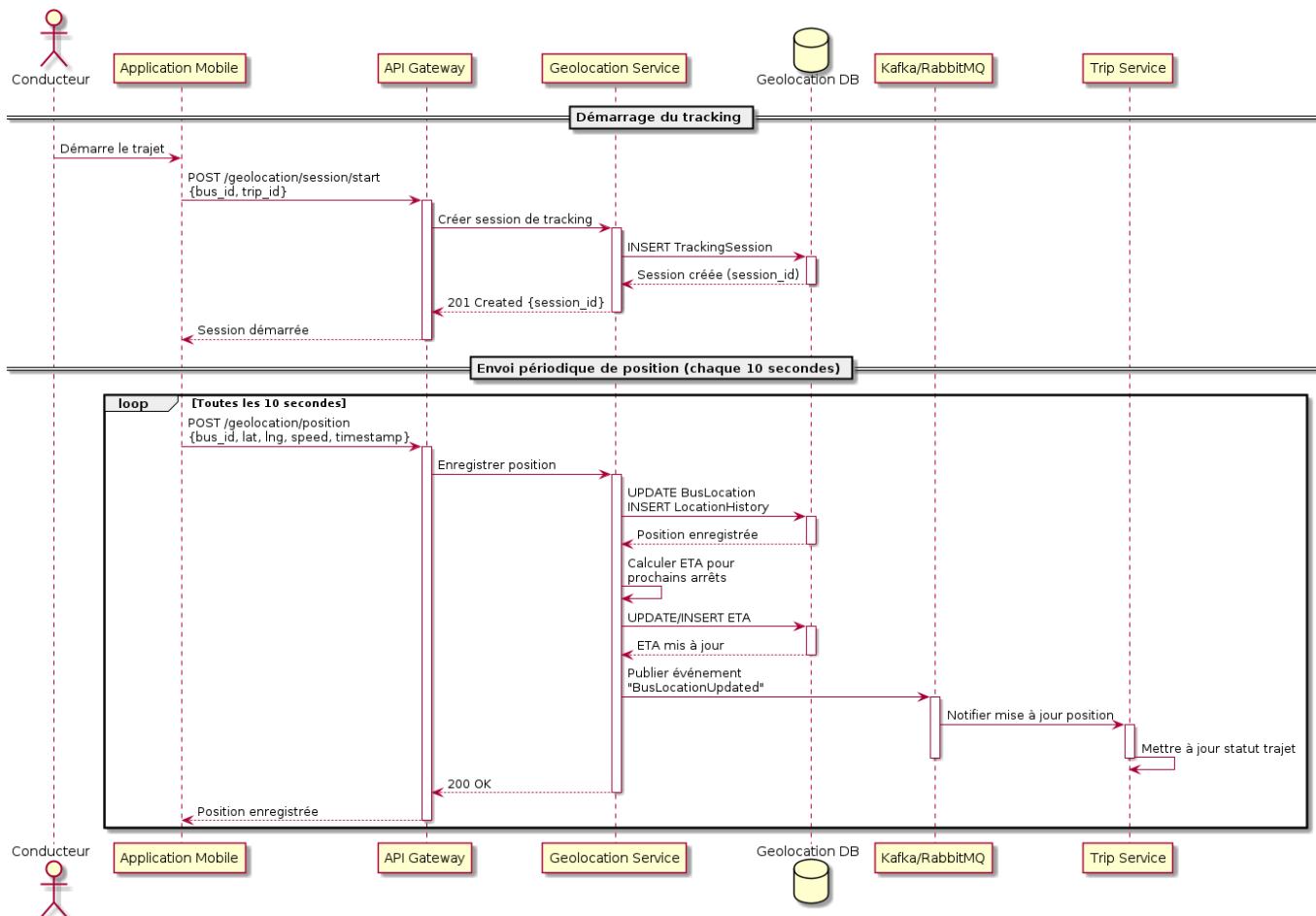


FIGURE 4.24 – Diagramme de séquence du processus de tracking en temps réel

#### 4.2.7 Service Notifications (Notification Service)

##### 4.2.7.1 Objectif

Gérer l'envoi de notifications multi-canaux (SMS, Email, Push) vers les utilisateurs. Ce service écoute les événements des autres services (achats, retards, annulations) et envoie des notifications selon les préférences de l'utilisateur. Il gère également les templates, règles de notification et historique.

##### 4.2.7.2 Cas d'utilisation

###### Principaux cas d'utilisation :

- Recevoir notification achat ticket
- Recevoir notification abonnement
- Recevoir alerte retard
- Recevoir alerte annulation
- Gérer préférences de notification
- Consulter historique notifications
- Envoyer notification SMS
- Envoyer notification Email
- Envoyer notification Push
- Créer template notification (Admin)
- Configurer règles de notification (Admin)

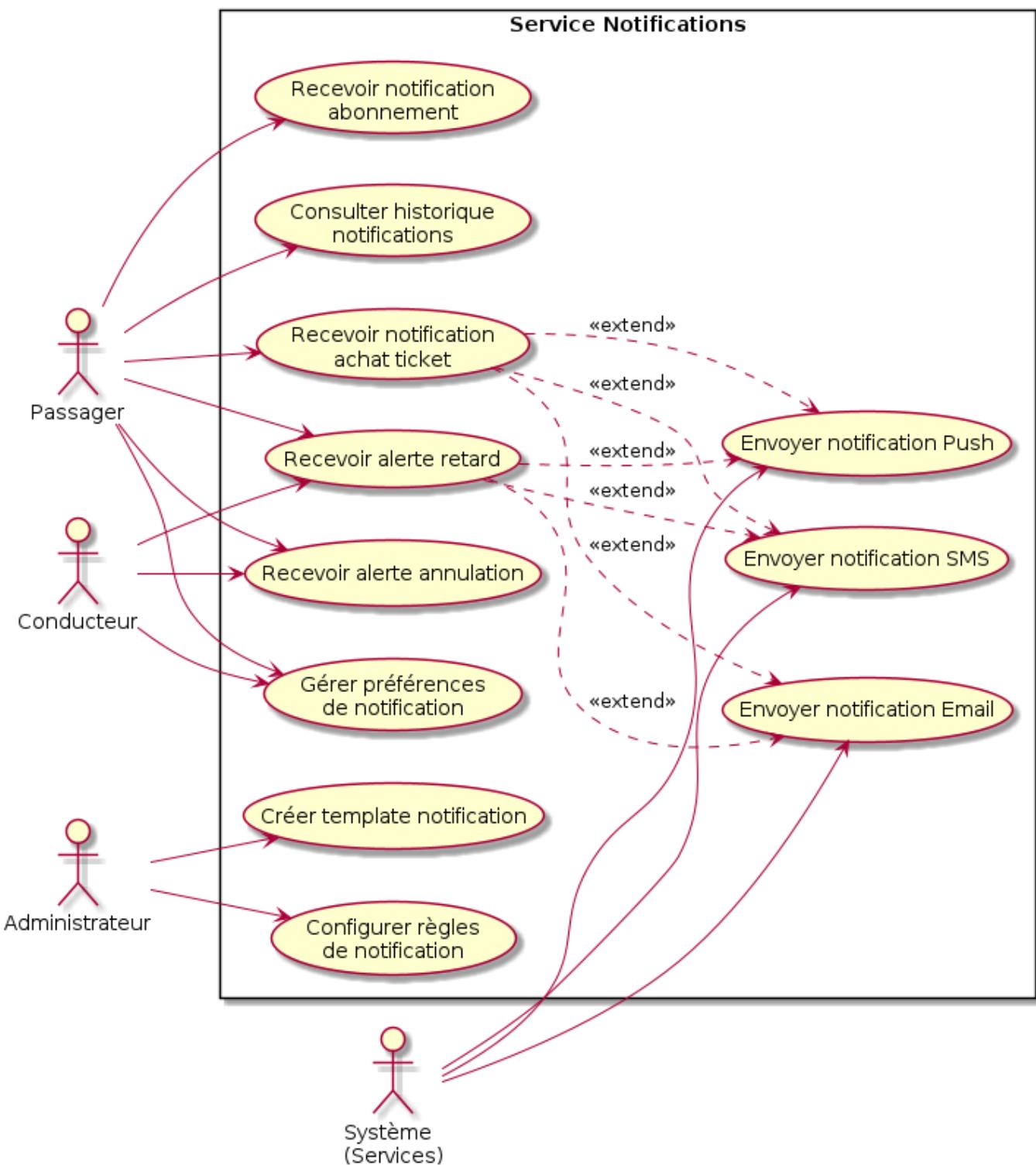


FIGURE 4.25 – Diagramme de cas d'utilisation du service Notifications

#### 4.2.7.3 Diagramme de classes

Classes principales pour Notification Service :

- **Notification** : notification principale (titre, message, canal, priorité, statut)
- **NotificationPreference** : préférences utilisateur par type et canal

- **NotificationTemplate** : templates réutilisables avec variables
- **NotificationRule** : règles d'envoi automatique basées sur événements
- **SMSNotification** : détails spécifiques aux SMS (numéro, fournisseur, coût)
- **EmailNotification** : détails spécifiques aux emails (adresse, HTML, suivi ouverture)
- **PushNotification** : détails spécifiques aux notifications push (device token, données)
- **NotificationLog** : historique et traçabilité des notifications

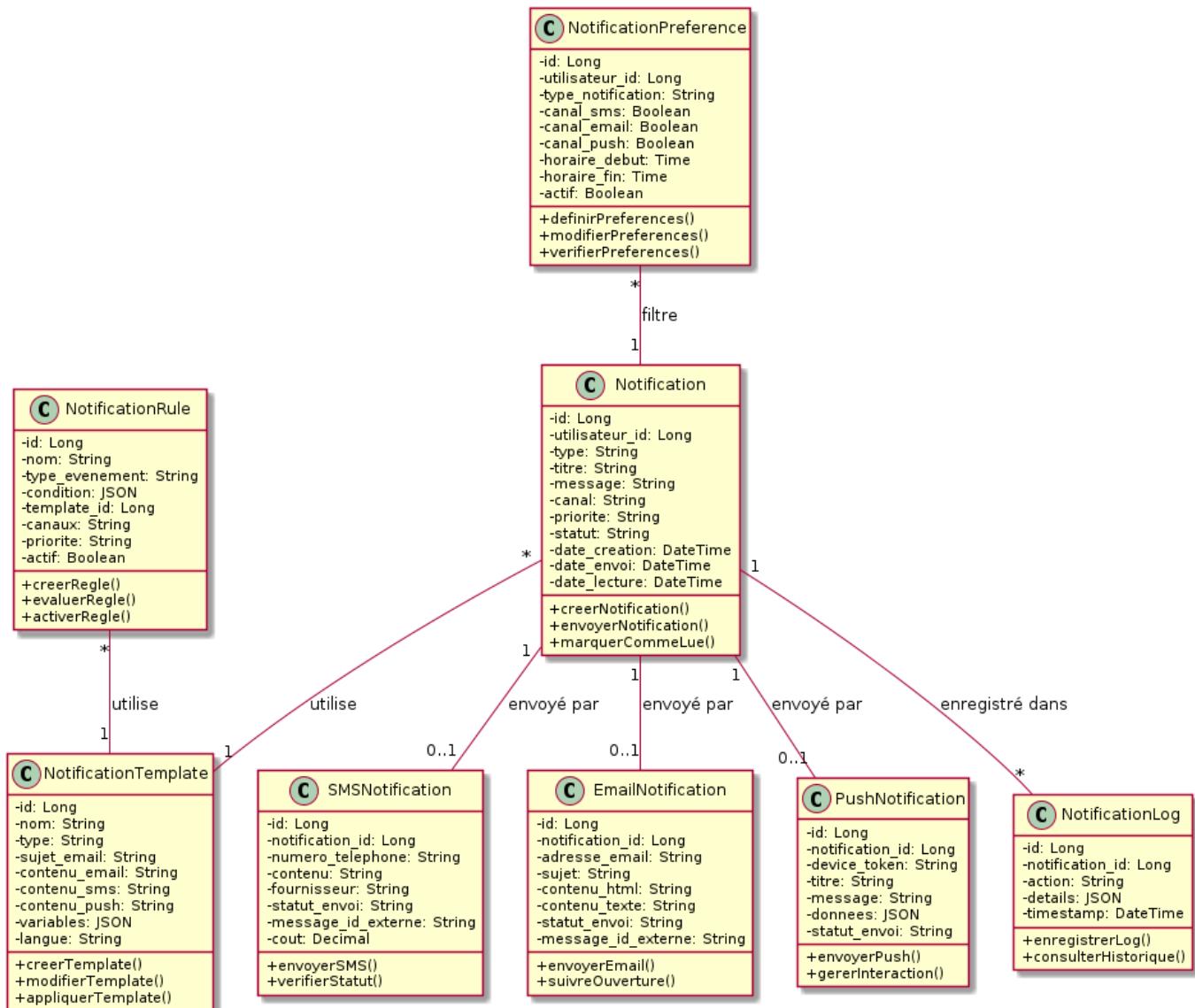


FIGURE 4.26 – Diagramme de classes du service Notifications

#### 4.2.7.4 Base de données et entités (Database per Service)

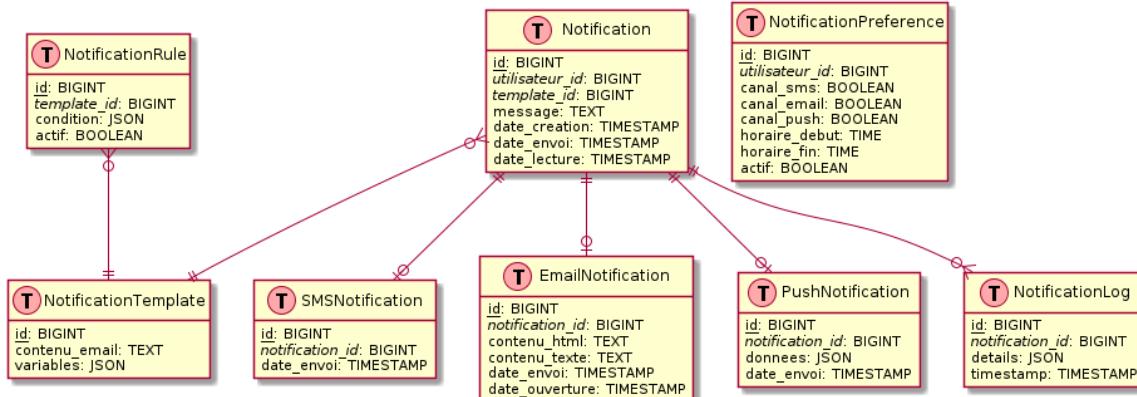


FIGURE 4.27 – Architecture de la base de données pour le service Notifications

Table	Champs
Notification	id (PK), utilisateur_id (FK), template_id (FK), type, titre, message, canal, priorite, statut, date_creation, date_envoi, date_lecture
NotificationPreference	id (PK), utilisateur_id (FK), type_notification, canal_sms, canal_email, canal_push, horaire_debut, horaire_fin, actif
NotificationTemplate	id (PK), nom, type, sujet_email, contenu_email, contenu_sms, contenu_push, variables (JSON), langue
NotificationRule	id (PK), template_id (FK), nom, type_evenement, condition (JSON), canaux, priorite, actif
SMSNotification	id (PK), notification_id (FK), numero_telephone, contenu, fournisseur, statut_envoi, message_id_externe, cout, date_envoi
EmailNotification	id (PK), notification_id (FK), adresse_email, sujet, contenu_html, contenu_texte, statut_envoi, message_id_externe, date_envoi, date_ouverture
PushNotification	id (PK), notification_id (FK), device_token, titre, message, donnees (JSON), statut_envoi, date_envoi
NotificationLog	id (PK), notification_id (FK), action, details (JSON), timestamp

#### 4.2.7.5 API REST / Endpoints

Endpoint	Méthode	Description
/notifications/user/{userId}	GET	Liste des notifications d'un utilisateur
/notifications/{id}	GET	Détails d'une notification
/notifications/{id}/read	PUT	Marquer comme lue

/notifications/send	POST	Envoyer une notification (interne)
/notifications/preferences/{userId}	GET	Préférences de notification d'un utilisateur
/notifications/preferences	PUT	Modifier préférences de notification
/notifications/templates	GET	Liste des templates (Admin)
/notifications/templates	POST	Créer un template (Admin)
/notifications/templates/{id}	PUT	Modifier un template (Admin)
/notifications/rules	GET	Liste des règles (Admin)
/notifications/rules	POST	Créer une règle (Admin)
/notifications/rules/{id}/activate	PUT	Activer/désactiver règle (Admin)
/notifications/history/{userId}	GET	Historique des notifications d'un utilisateur
/notifications/stats	GET	Statistiques d'envoi (Admin)

#### 4.2.7.6 Diagramme de séquence – Envoi de notification de retard

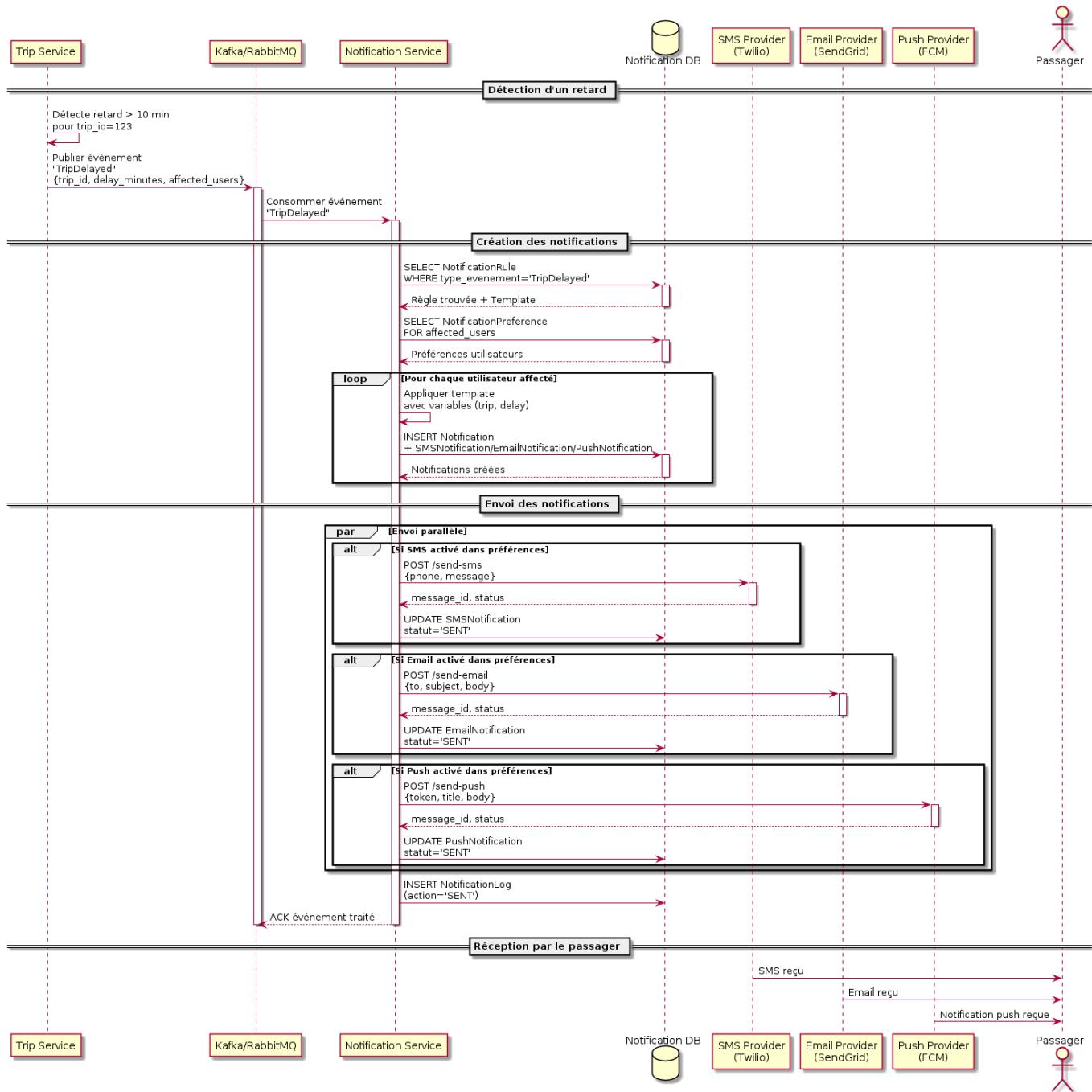


FIGURE 4.28 – Diagramme de séquence du processus d'envoi de notification de retard

#### 4.2.8 Service Payment (Payment Service)

##### 4.2.8.1 Objectif

Gérer le traitement des paiements pour l'achat de tickets et le renouvellement d'abonnements. Ce service s'intègre avec des passerelles de paiement externes (Stripe, PayPal, etc.), génère des factures, gère les remboursements et conserve l'historique complet des transactions pour la conformité et l'audit.

#### 4.2.8.2 Cas d'utilisation

**Principaux cas d'utilisation :**

- Acheter un ticket
- Renouveler un abonnement
- Consulter historique des paiements
- Demander un remboursement
- Enregistrer méthode de paiement
- Supprimer méthode de paiement
- Traiter le paiement (Passerelle)
- Valider le paiement (Passerelle)
- Générer facture
- Traiter remboursement (Admin)
- Gérer les transactions (Admin)
- Configurer moyens de paiement (Admin)

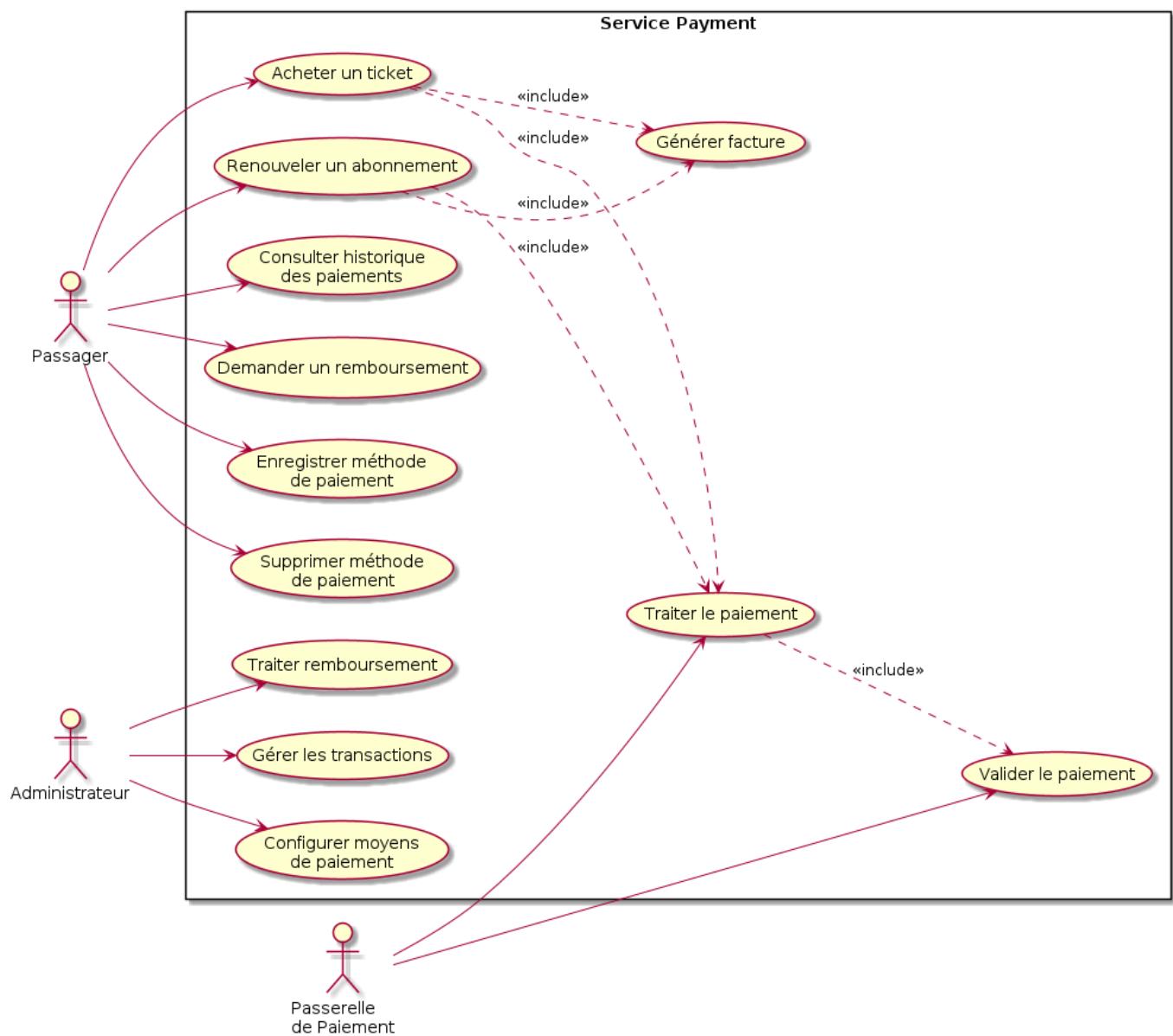
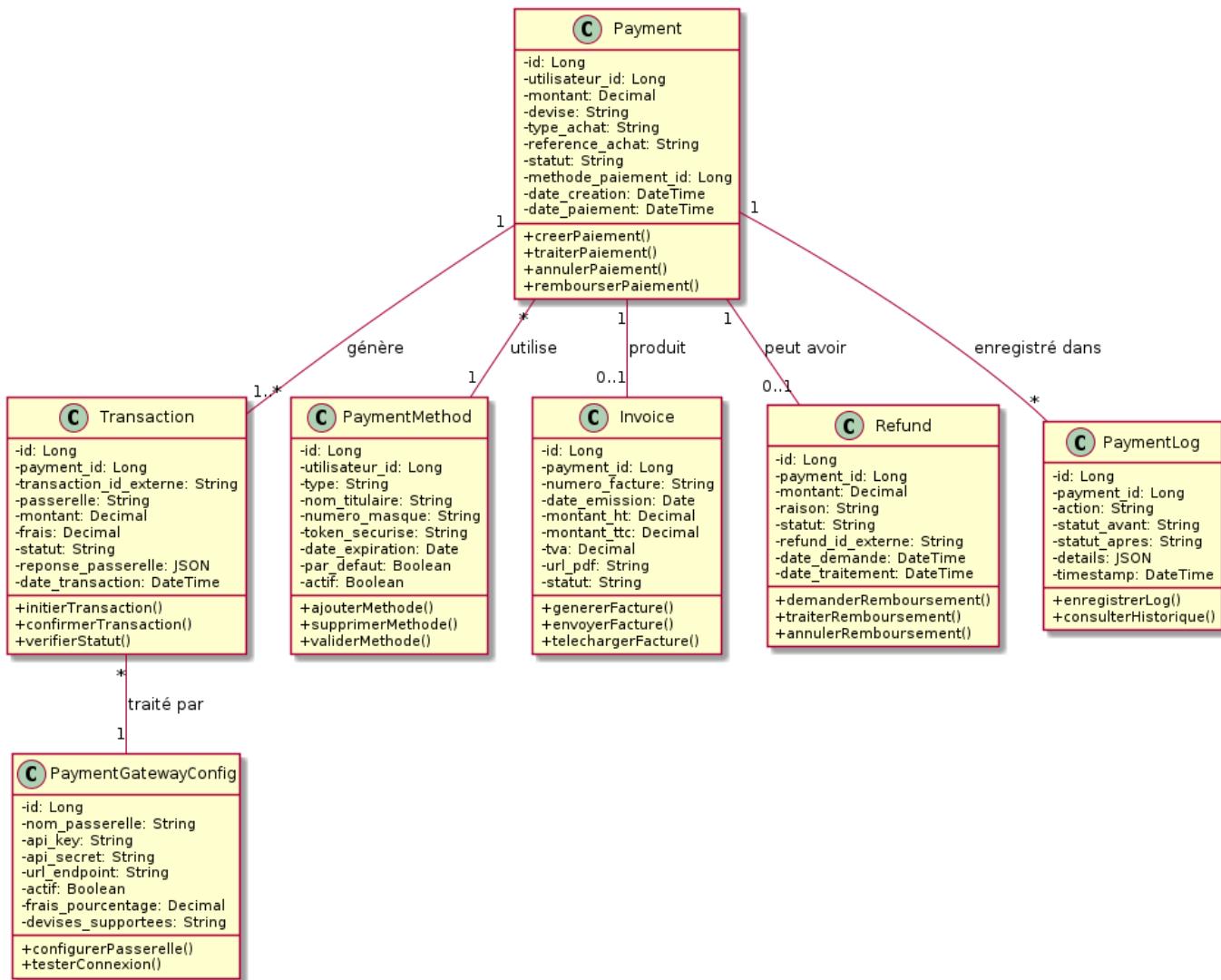


FIGURE 4.29 – Diagramme de cas d'utilisation du service Payment

#### 4.2.8.3 Diagramme de classes

Classes principales pour Payment Service :

- **Payment** : paiement principal (montant, devise, type d'achat, statut)
- **Transaction** : détails de la transaction avec la passerelle externe
- **PaymentMethod** : méthode de paiement enregistrée (carte, PayPal, etc.)
- **Invoice** : facture générée (numéro, montants HT/TTC, PDF)
- **Refund** : remboursement (montant, raison, statut)
- **PaymentGatewayConfig** : configuration des passerelles de paiement
- **PaymentLog** : historique et traçabilité des paiements



#### 4.2.8.4 Base de données et entités (Database per Service)

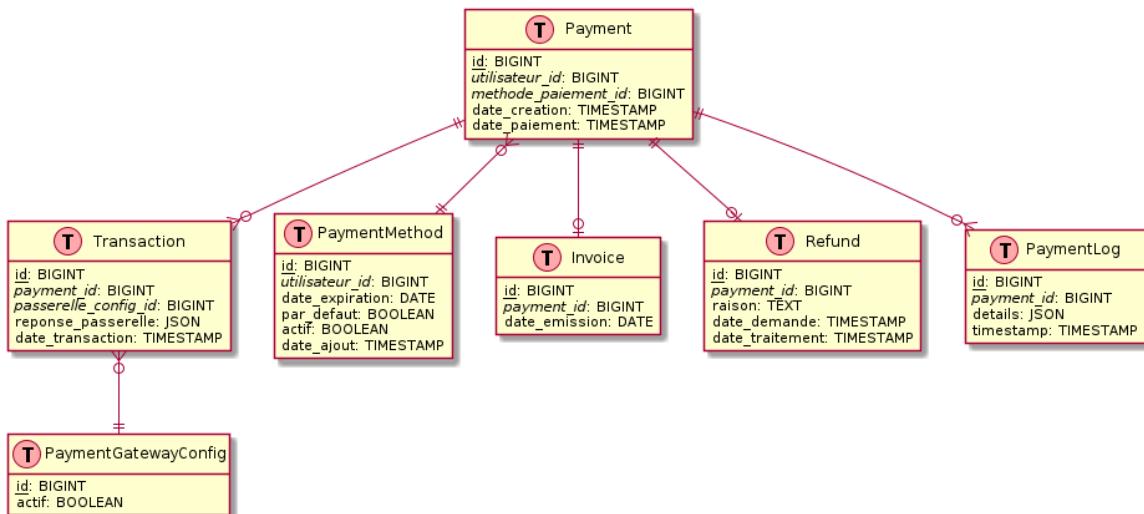


FIGURE 4.31 – Architecture de la base de données pour le service Payment

Table	Champs
Payment	id (PK), utilisateur_id (FK), methode_paiement_id (FK), montant, devise, type_achat, reference_achat, statut, date_creation, date_paiement
Transaction	id (PK), payment_id (FK), passerelle_config_id (FK), transaction_id_externe, passerelle, montant, frais, statut, reponse_passerelle (JSON), date_transaction
PaymentMethod	id (PK), utilisateur_id (FK), type, nom_titulaire, numero_masque, token_securise, date_expiration, par_defaut, actif, date_ajout
Invoice	id (PK), payment_id (FK), numero_facture, date_emission, montant_ht, montant_ttc, tva, url_pdf, statut
Refund	id (PK), payment_id (FK), montant, raison, statut, refund_id_externe, date_demande, date_traitement
PaymentGatewayConfig	id (PK), nom_passerelle, api_key, api_secret, url_endpoint, actif, frais_pourcentage, devises_supportees
PaymentLog	id (PK), payment_id (FK), action, statut_avant, statut_apres, details (JSON), timestamp

#### 4.2.8.5 API REST / Endpoints

Endpoint	Méthode	Description
/payment/create	POST	Créer un nouveau paiement

/payment/{id}	GET	Détails d'un paiement
/payment/confirm	POST	Confirmer un paiement après validation passerelle
/payment/fail	POST	Marquer un paiement comme échoué
/payment/user/{userId}/history	GET	Historique des paiements d'un utilisateur
/payment/methods/{userId}	GET	Méthodes de paiement enregistrées
/payment/methods	POST	Ajouter une méthode de paiement
/payment/methods/{id}	DELETE	Supprimer une méthode de paiement
/payment/invoice/{paymentId}	GET	Télécharger la facture d'un paiement
/payment/refund	POST	Demander un remboursement
/payment/refund/{id}	GET	Statut d'un remboursement
/payment/refund/{id}/process	POST	Traiter un remboursement (Admin)
/payment/gateways	GET	Liste des passerelles configurées (Admin)
/payment/gateways	POST	Ajouter une passerelle (Admin)
/payment/transactions	GET	Liste des transactions (Admin)
/payment/stats	GET	Statistiques de paiement (Admin)

#### 4.2.8.6 Diagramme de séquence – Achat d'un ticket

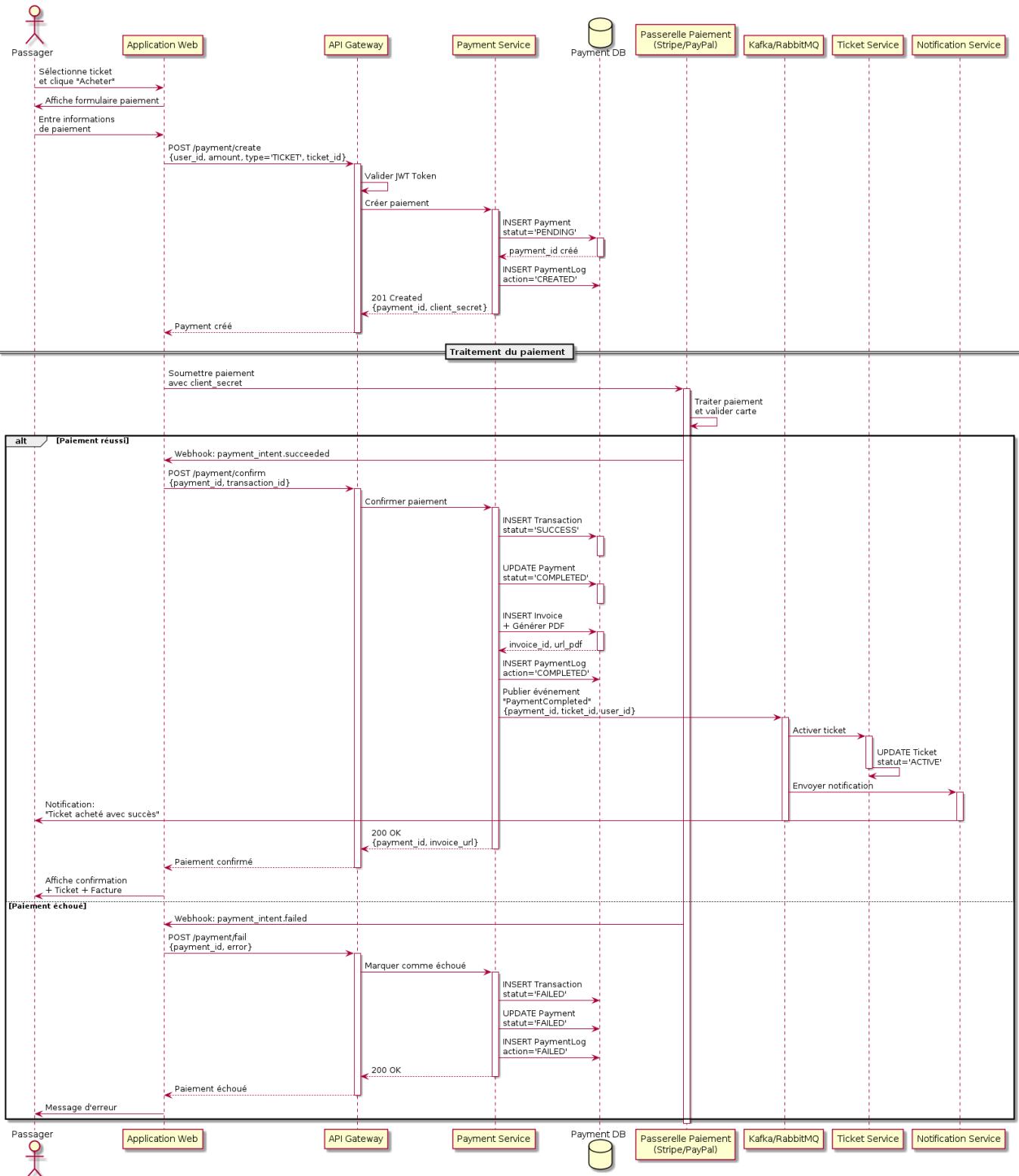


FIGURE 4.32 – Diagramme de séquence du processus d'achat de ticket avec paiement

## 4.3 Conclusion

La conception détaillée fournit toutes les spécifications techniques nécessaires au développement. Les diagrammes de classes, séquences et schémas de bases de données définissent une architecture robuste et cohérente. Cette modélisation complète garantit l'autonomie des services tout en assurant leur interopérabilité, posant ainsi les bases techniques solides pour la phase d'implémentation.