Rapport Conception du projet SOA/Microservices

Filière : Génie logiciel

Conception et Développement d'un Système de Gestion basé sur une Architecture Microservices pour une Entreprise de Transport Urbain

---

Réalisé par :

AKOUR Ayoub
ETTALBI Omar
AIT TALEB Saad

Encadré par :

Pr. Mahmoud Nassar

Année Universitaire 2024-2025

LISTE DE FIGURES

TABLE DES MATIÈRES

CHAPITRE 1

CONCEPTION DE L'APPLICATION

1.1 Diagramme de Cas d'Utilisation

1.1.1 Diagramme de Cas d'Utilisation : Authentification et Gestion du Compte

Ce diagramme de cas d'utilisation décrit les interactions liées à l'authentification et à la gestion du compte utilisateur. Il montre les actions accessibles à un visiteur non authentifié (inscription et connexion) ainsi que les opérations disponibles pour un passager connecté, telles que la gestion du profil, la modification des informations personnelles ou la déconnexion. Il met également en évidence l'importance du mécanisme d'authentification JWT pour sécuriser l'accès aux fonctionnalités de gestion de compte.

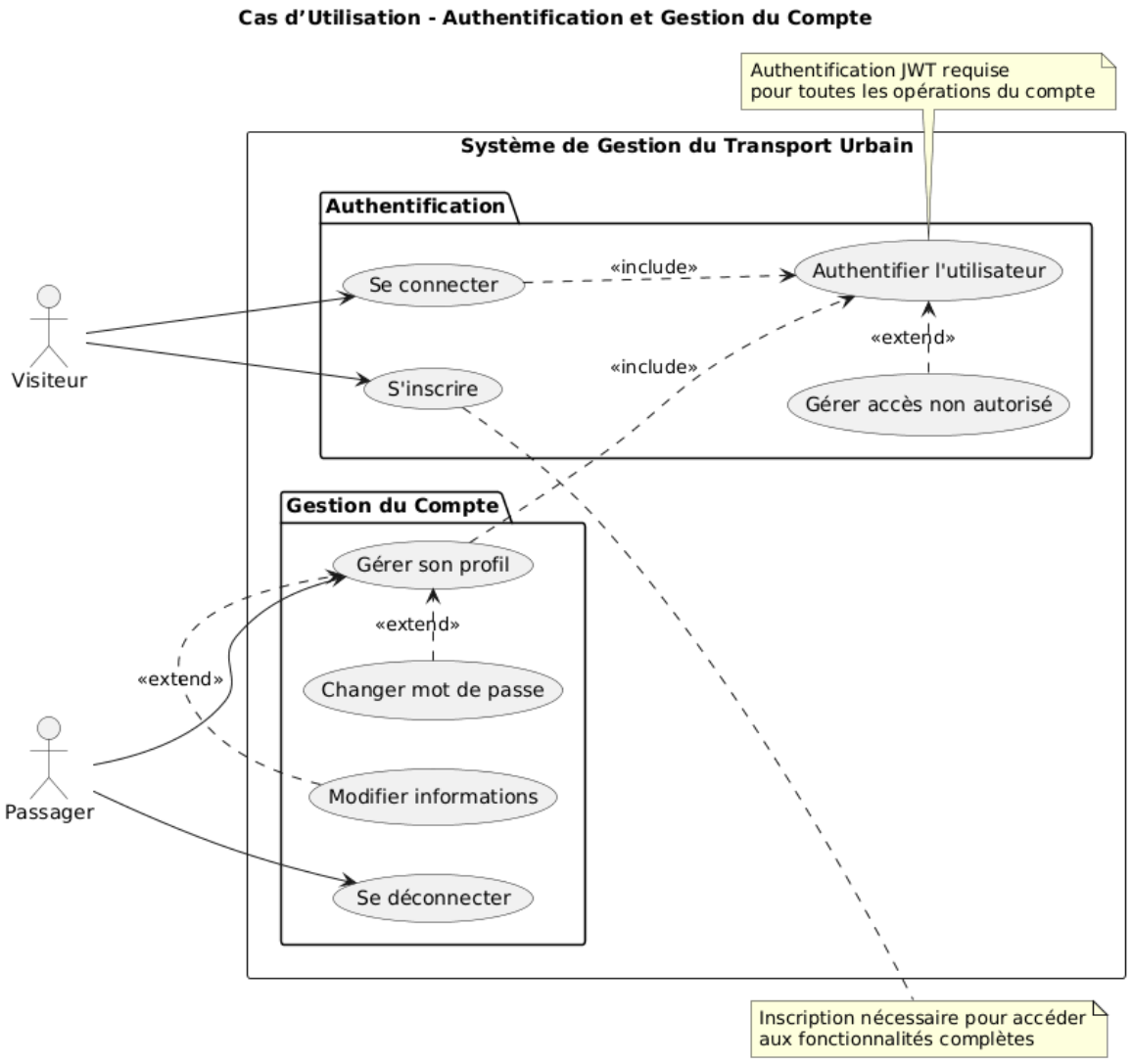

FIGURE 1.1 – Authentication & Account Management use case diagram

1.1.2 Diagramme de Cas d'Utilisation : Gestion des Tickets

Ce diagramme illustre les différents cas d'utilisation relatifs à l'achat et à la gestion des tickets de transport. Le passager peut acheter un ticket, consulter son historique, annuler un ticket ou télécharger son QR code. Le diagramme met aussi en évidence l'interaction avec le système de paiement pour la finalisation des transactions et le service de notification pour informer l'utilisateur après chaque achat ou annulation. Il met en avant le flux complet de l'expérience d'achat, de la sélection du ticket jusqu'à la réception de la confirmation.

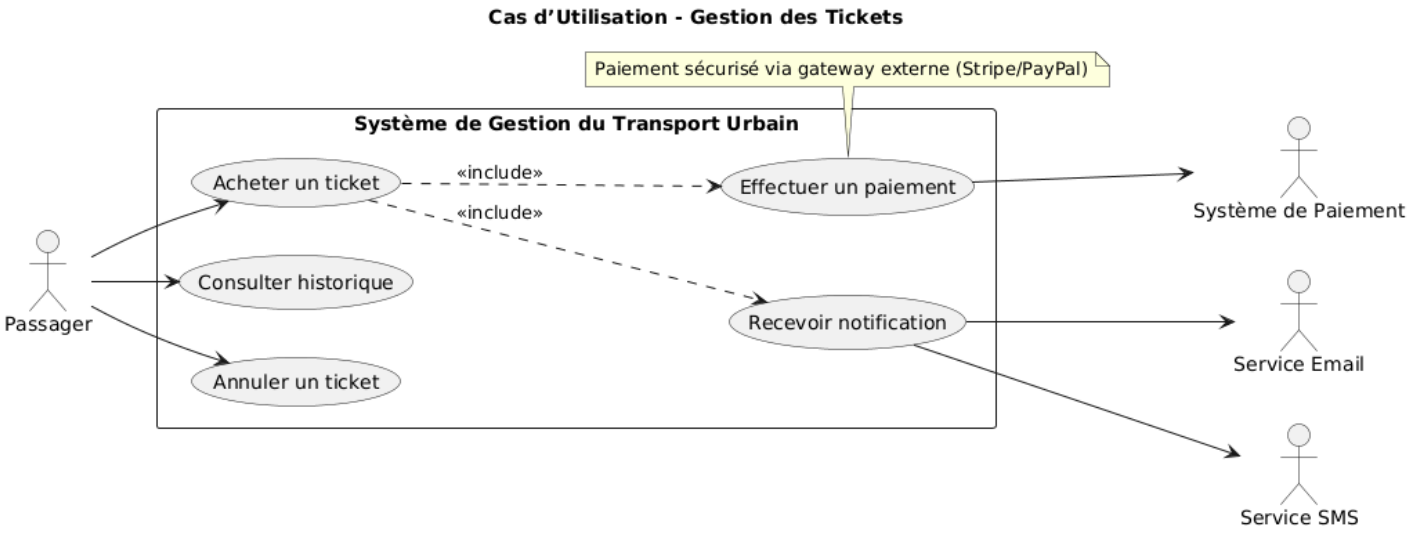

FIGURE 1.2 – Ticket Management use case diagram

1.1.3 Diagramme de Cas d'Utilisation : Gestion des Abonnements

Ce diagramme présente les cas d'utilisation liés à la souscription, au renouvellement et à l'annulation des abonnements. Le passager peut souscrire un abonnement, le renouveler ou le résilier à tout moment via l'interface utilisateur. Le système interagit avec le service de paiement pour la gestion des transactions et avec le service de notification pour informer l'utilisateur des changements d'état de son abonnement (renouvellement automatique, expiration ou annulation). Ce diagramme met en évidence la gestion continue et automatisée des abonnements, garantissant une expérience fluide et sans interruption pour les usagers réguliers.

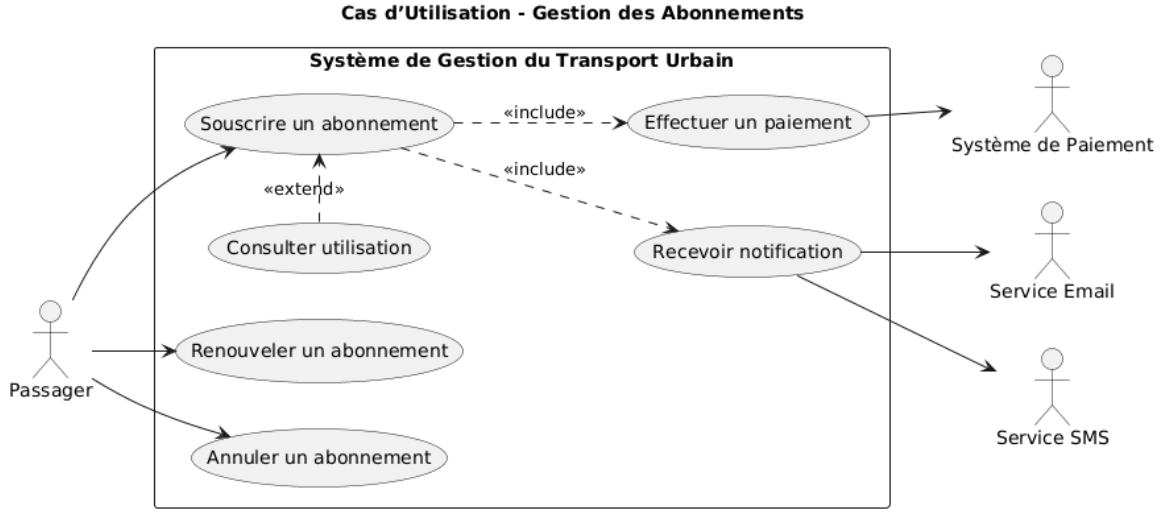

FIGURE 1.3 – Subscription Management use case diagram

1.1.4 Diagramme de Cas d'Utilisation : Consultation et Suivi des Trajets

Ce diagramme décrit les interactions permettant aux utilisateurs de consulter les horaires, les trajets disponibles et de suivre en temps réel la position des bus. Le passager ou le visiteur peut

rechercher un itinéraire, afficher les horaires, et calculer un trajet optimal grâce à l'intégration avec une API de géolocalisation (Google Maps, OpenStreetMap, etc.). Cette vue d'ensemble met en évidence la dimension dynamique du système et son rôle dans l'amélioration de l'expérience utilisateur grâce à des informations actualisées en temps réel.

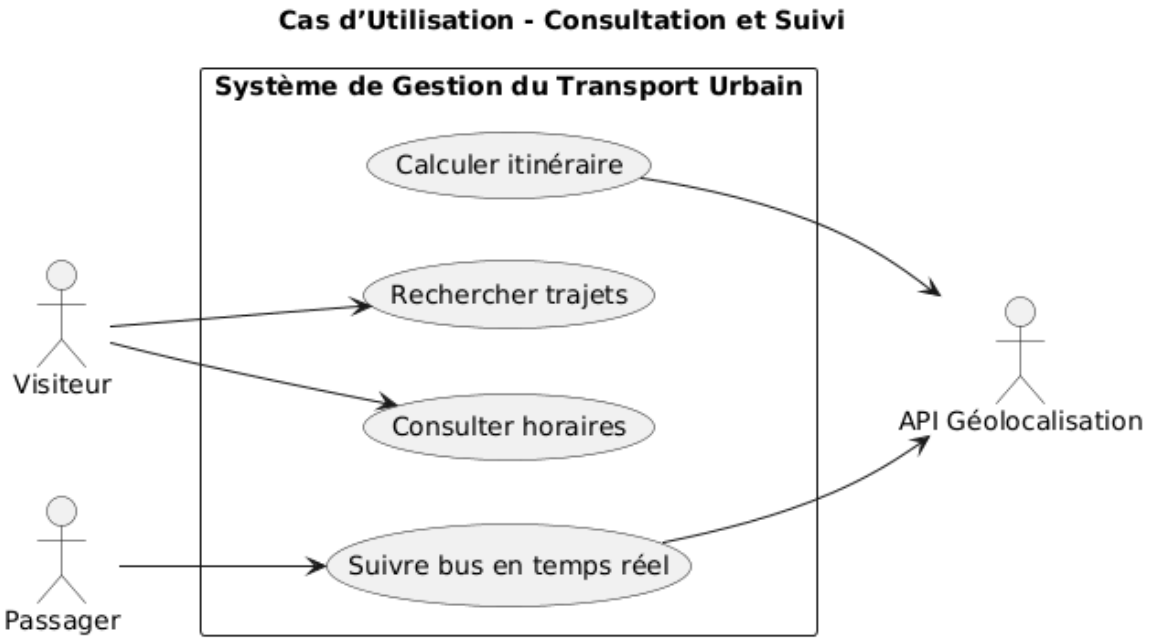

FIGURE 1.4 – Route Consultation and Tracking use case diagram

1.1.5 Diagramme de Cas d'Utilisation : Notifications

Ce diagramme illustre le fonctionnement du système de notifications qui informe les utilisateurs en cas de retards, d'annulations, ou de changements d'itinéraires. Le passager peut configurer ses préférences pour recevoir les alertes par e-mail ou SMS. Le système interagit avec des services externes de messagerie et de télécommunication pour assurer une communication fiable et rapide. Cette composante joue un rôle essentiel dans la réactivité et la satisfaction des usagers du transport urbain.

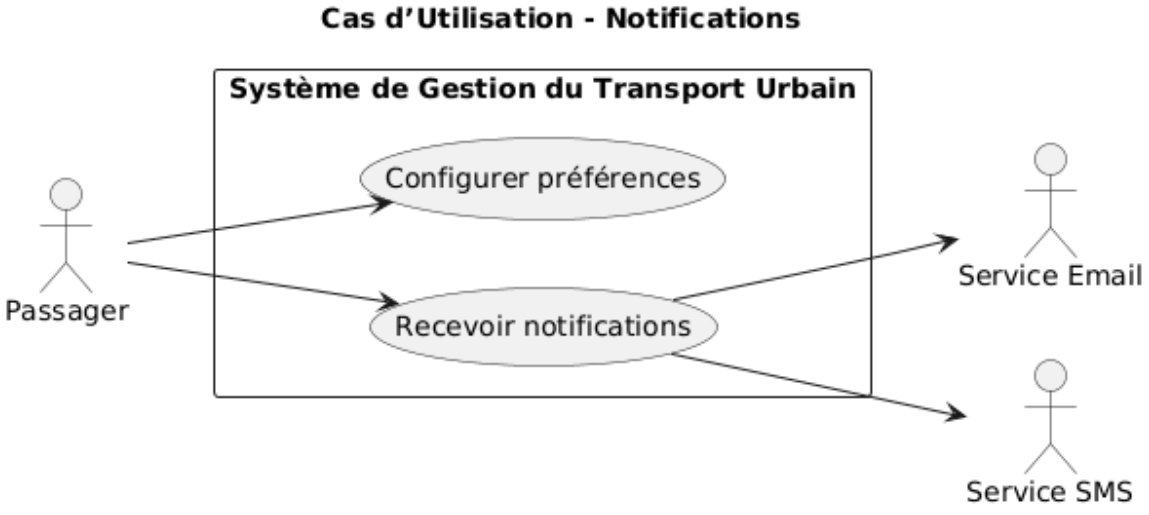

FIGURE 1.5 – Notification Management use case diagram

1.2 Diagramme de Classes

Le diagramme de classes illustre la structure statique du système de gestion du transport urbain. Il met en évidence les principales entités métier (utilisateurs, trajets, tickets, abonnements, bus, etc.), leurs attributs, leurs méthodes, ainsi que les relations d'héritage et d'association qui les lient. Ce diagramme constitue la base de la modélisation orientée objet du système et facilite la compréhension de son organisation logique.

User : représente un utilisateur du système, qu'il soit passager ou conducteur. Elle contient les informations d'authentification et de profil (*id, nom, email, mot de passe, rôle, état du compte*). Elle définit les opérations liées à la connexion, la déconnexion et la mise à jour du profil utilisateur. Cette classe constitue la classe mère pour les entités *Passenger* et *Driver*.

Passenger : hérite de la classe *User* et représente les usagers du service de transport. Chaque passager possède un solde de points de fidélité et peut acheter des tickets ou souscrire à des abonnements. Cette classe est liée à plusieurs entités, notamment *Ticket* et *Subscription*, qu'un passager peut acquérir.

Driver : hérite également de la classe *User* et représente les conducteurs. Chaque conducteur possède un numéro de licence et une date d'embauche, et il est associé à un ou plusieurs bus via une relation « conduit ». Les méthodes *startShift()* et *endShift()* permettent de gérer le début et la fin de leurs services.

Bus : cette classe représente un véhicule du réseau de transport. Chaque bus possède un identifiant, un numéro d'immatriculation, une capacité et une position actuelle. Il est conduit par un conducteur et est assigné à une route spécifique (relation « suit »). La méthode *updateLocation()* permet de mettre à jour la position du bus en temps réel.

Route : correspond à un itinéraire reliant une origine et une destination avec une distance associée. Une route peut avoir plusieurs horaires planifiés (relation « une route a plusieurs schedules »). Elle permet ainsi de définir les trajets réguliers du réseau.

Schedule : représente un horaire de trajet pour une route donnée. Il contient les informations de départ, d'arrivée et permet de calculer la durée du trajet via la méthode *calculateDuration()*. Chaque horaire est lié à un trajet (*Route*) et à un ticket via la relation « correspond à ».

Ticket : représente un titre de transport acheté par un passager. Un ticket est associé à un horaire spécifique (*Schedule*) et possède un prix, une date d'émission et un statut (valide, annulé, expiré, etc.). Le passager peut valider ou annuler un ticket grâce aux méthodes *validate()* et *cancel()*.

Subscription : décrit un abonnement souscrit par un passager pour une période donnée (mensuelle ou annuelle). Chaque abonnement a un type, un prix et une période de validité. Il est lié à un passager (relation « un passager peut avoir plusieurs abonnements »). Les méthodes *renew()* et *cancel()* permettent de gérer le cycle de vie de l'abonnement.

Notification : gère l'envoi de messages d'information aux utilisateurs, tels que les retards, les annulations ou les confirmations d'achat. Elle contient le message, la date d'envoi et une méthode *sendToUser()* pour notifier l'utilisateur. Chaque notification peut être envoyée à un ou plusieurs utilisateurs.

1.2.1 Relations entre les classes

— **Héritage :** Les classes *Passenger* et *Driver* héritent de la classe *User*, illustrant la spécialisation des types d'utilisateurs.
— **Association :** Un *Driver* peut conduire plusieurs *Bus*, mais chaque *Bus* est associé à un seul *Driver*.
— **Composition :** Une *Route* possède plusieurs *Schedules*, ce qui représente les différents horaires liés à un trajet.
— **Association multiple :** Un *Passenger* peut acheter plusieurs *Tickets* et souscrire à plusieurs *Subscriptions*.
— **Dépendance :** Les *Notifications* dépendent des *Users* auxquels elles sont envoyées.

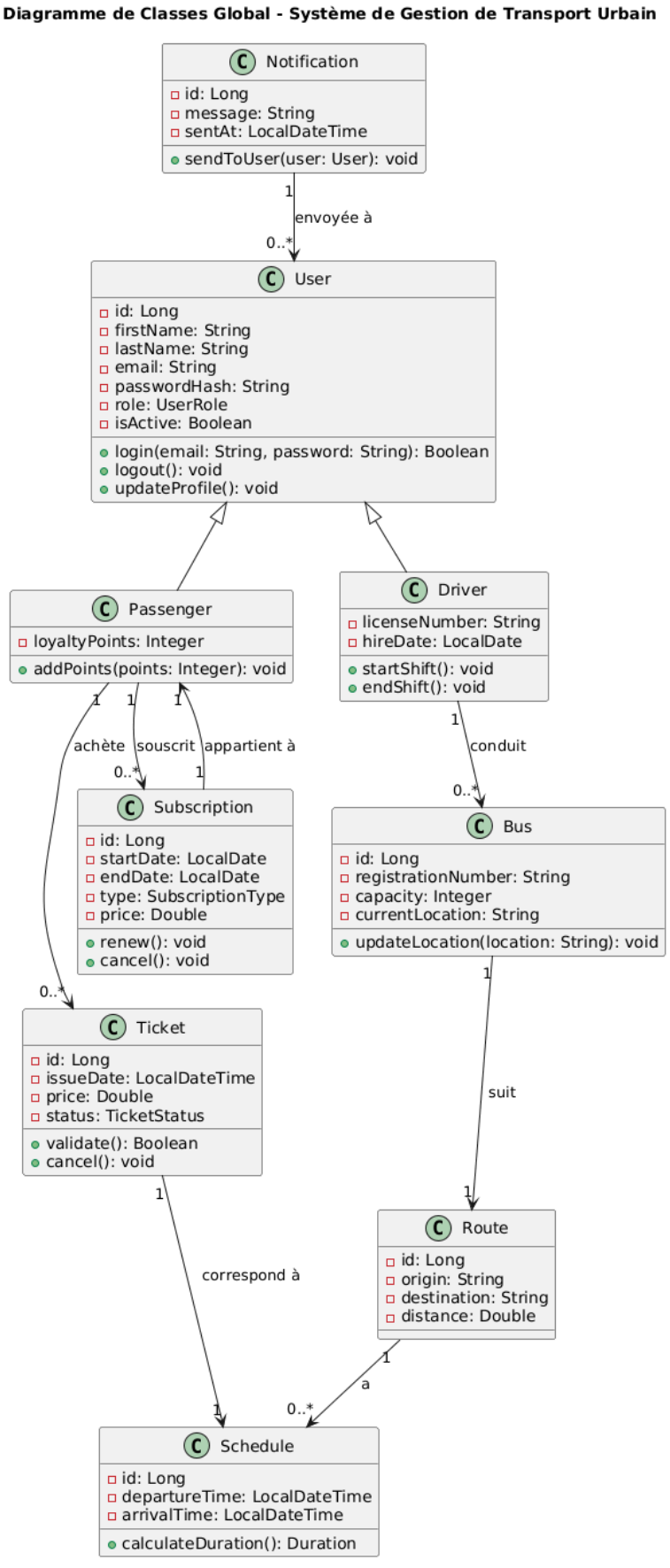

FIGURE 1.6 – Diagramme de Classes

1.3 Diagramme Entité–Association (Entity–Relation)

Cette section présente la conception des bases de données de chaque microservice du système de gestion de transport urbain. Conformément à l'architecture microservices, chaque service dispose de sa propre base de données afin de garantir l'isolation, la scalabilité et la cohérence locale (principe *Database per Service*).

1.3.1 User Service Database

Objectif : Gérer l'authentification, les profils utilisateurs et la gestion des rôles. Ce service stocke les informations des utilisateurs, y compris les passagers, conducteurs et administrateurs. Les relations entre les entités *users*, *passengers*, *drivers* et *admins* permettent de gérer des profils spécialisés à partir d'une entité commune.

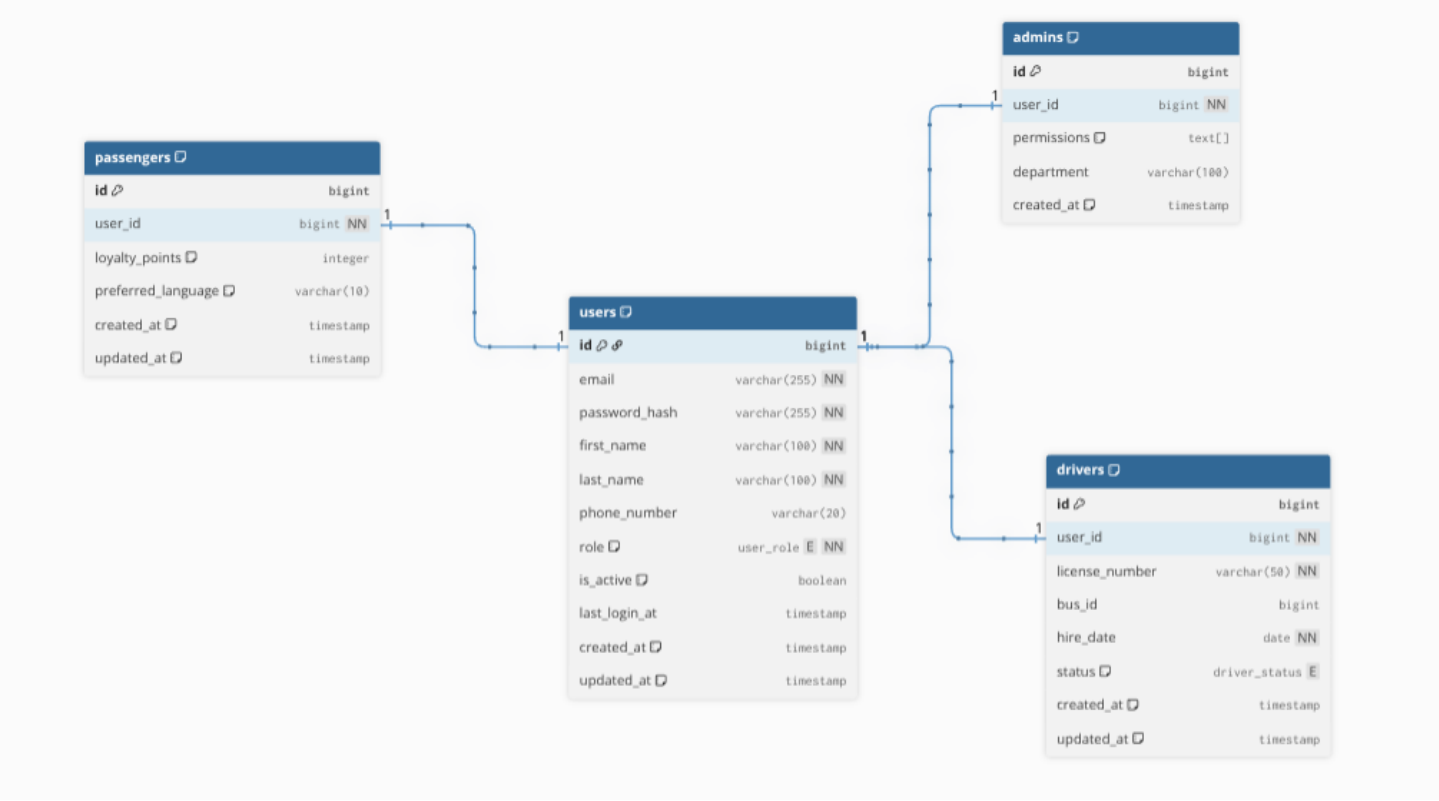

FIGURE 1.7 – Schéma entité–association du User Service

Points clés :

— Gestion centralisée des utilisateurs avec rôles distincts.
— Relations 1–1 entre *users* et ses extensions : *passengers*, *drivers*, *admins*.
— Enums définissant le rôle et le statut des conducteurs.

1.3.2 Ticket Service Database

Objectif : Gérer la commande, la vente et le paiement des tickets. Chaque commande peut contenir plusieurs tickets, mais un seul paiement est associé à une commande. Les remboursements sont gérés séparément pour garantir la traçabilité financière.

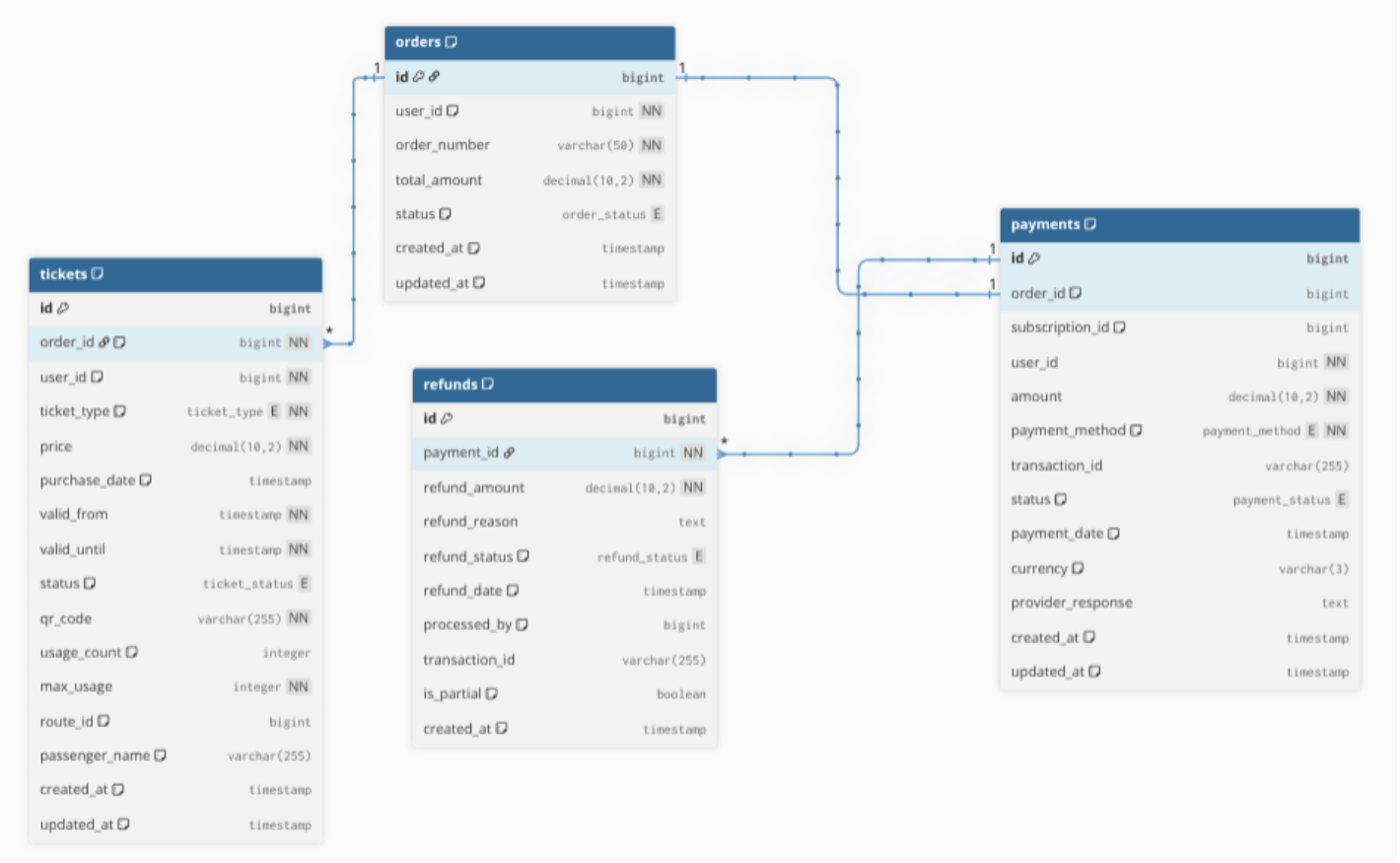

FIGURE 1.8 – Schéma entité–association du Ticket Service

Points clés :

— Relation 1–N entre *orders* et *tickets*.

— Paiement unique par commande (*payments.order_id* → *orders.id*).

— Remboursement possible complet ou partiel (*refunds.payment_id* → *payments.id*).

1.3.3 Route Service Database

Objectif : Gérer les routes, arrêts et horaires de bus. Ce service permet de définir les trajets du réseau ainsi que les horaires associés à chaque ligne.

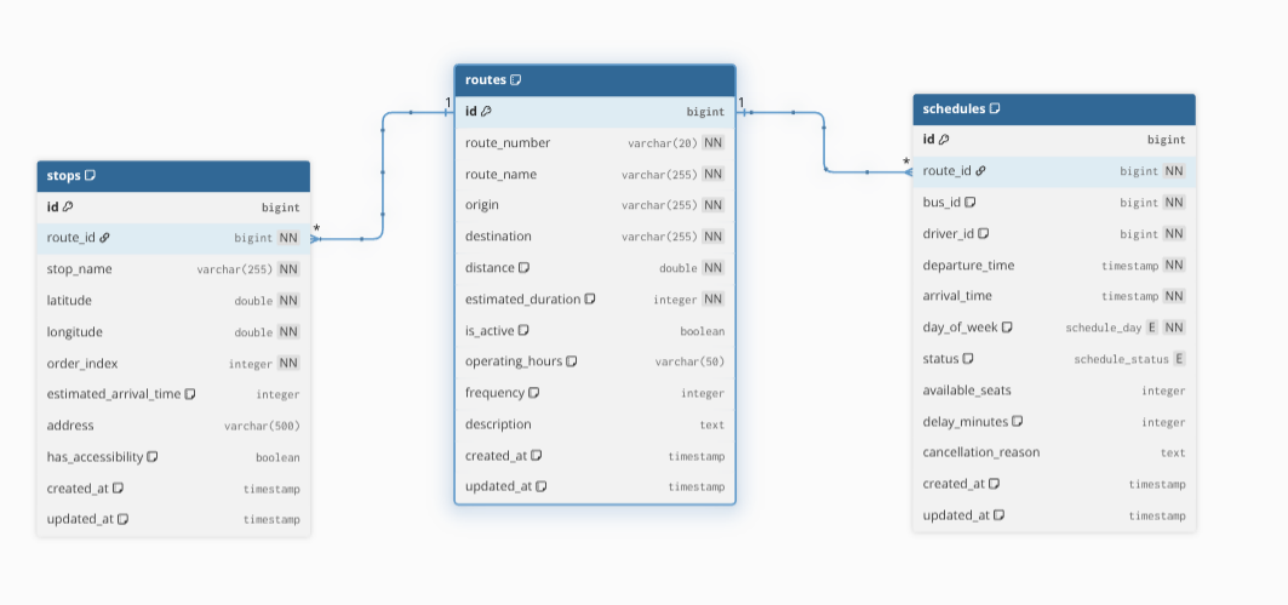

FIGURE 1.9 – Schéma entité–association de Route Service

Points clés :

— Relation 1–N entre *routes* et *stops*.
— Relation 1–N entre *routes* et *schedules*.
— Gestion du statut et du jour d'exécution d'un horaire via des énumérations.

1.3.4 Geolocation Service Database

Objectif : Suivre en temps réel la position et l'état des bus. Ce service conserve la position GPS, la vitesse et l'état de maintenance de chaque véhicule.

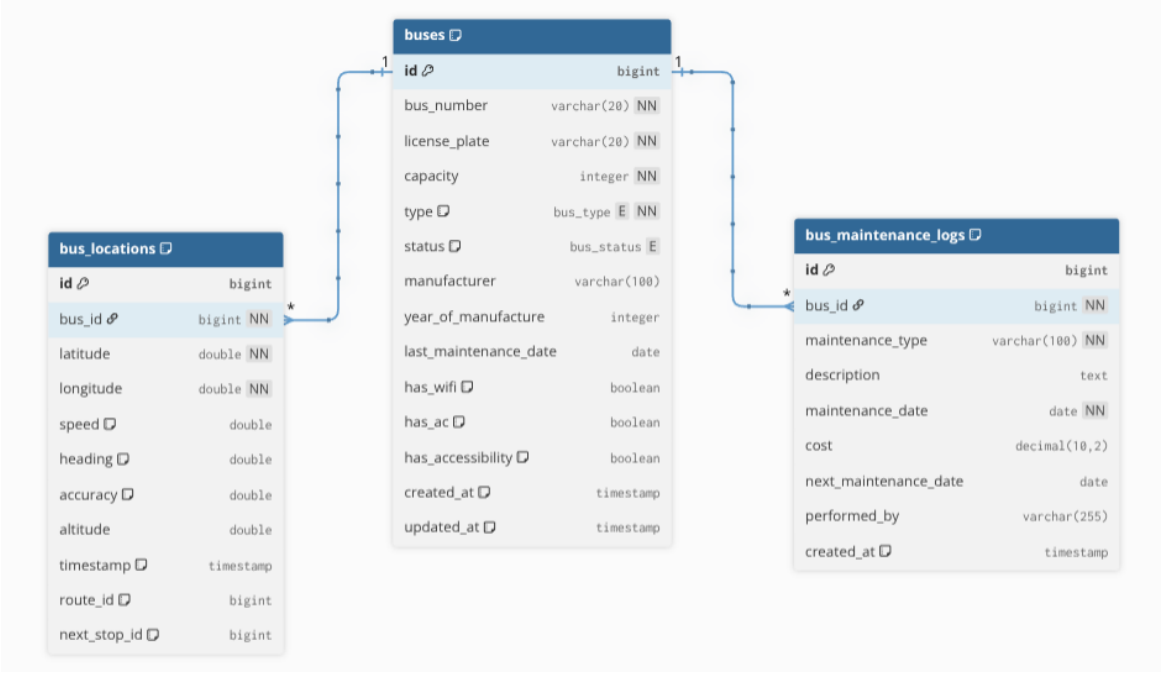

FIGURE 1.10 – Schéma entité–association du Geolocation Service

Points clés :

— Relation 1–N entre *buses* et *bus_ locations*.
— Historique des maintenances stocké dans *bus_ maintenance_ logs*.
— Champs pour vitesse, direction et précision GPS pour analyses temps réel.

1.3.5 Subscription Service Database

Objectif : Gérer les abonnements mensuels et annuels. Chaque abonnement appartient à un utilisateur et peut être associé à un paiement.

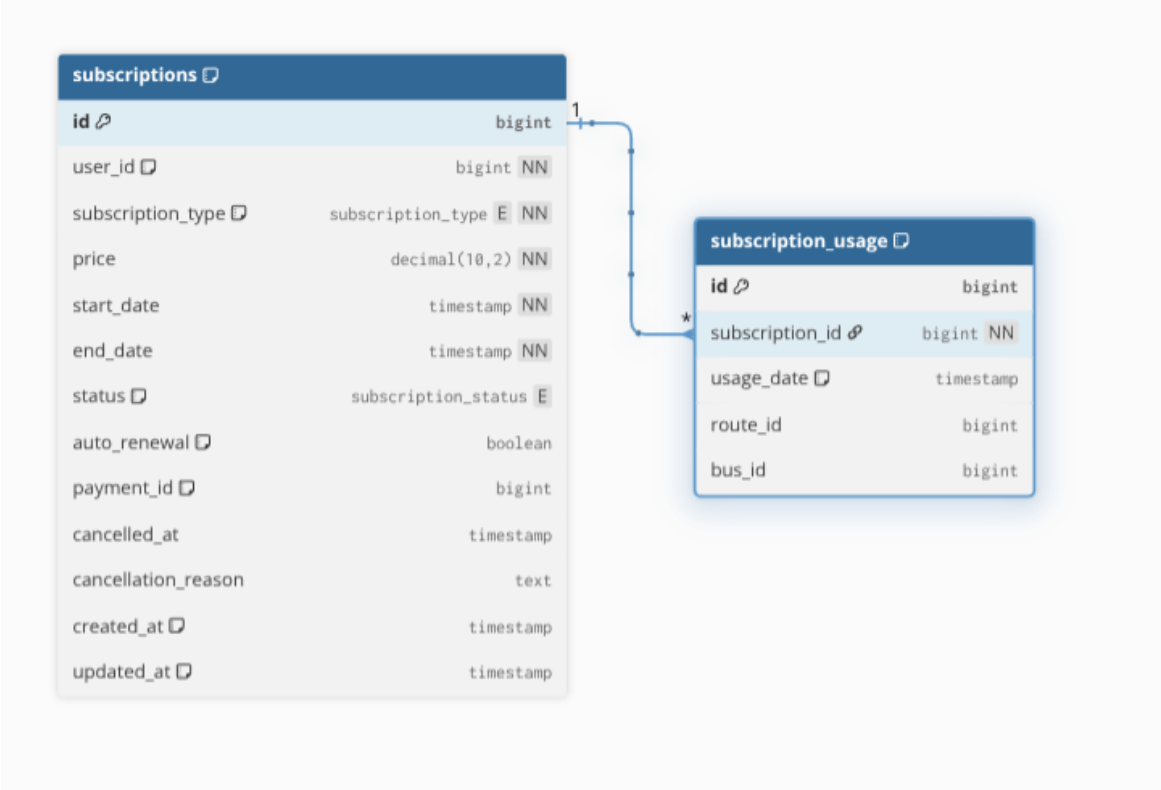

FIGURE 1.11 – Schéma entité–association du Subscription Service

Points clés :

— Relation 1–N entre *subscriptions* et *subscription_ usage*.
— Gestion du statut, du type et du renouvellement automatique.
— Historique d'utilisation pour suivi statistique.

1.3.6 Notification Service Database

Objectif : Gérer l'envoi de notifications aux utilisateurs par email, SMS ou push. Les préférences de notification sont stockées individuellement pour chaque utilisateur.

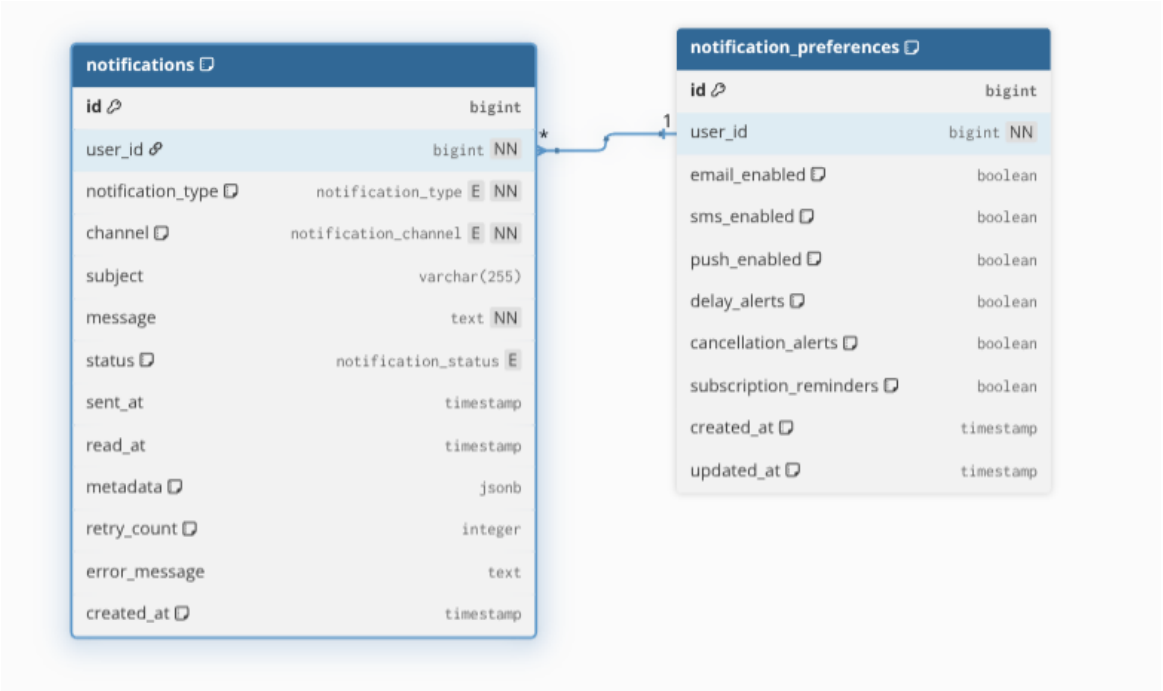

FIGURE 1.12 – Schéma entité–association du Notification Service

Points clés :

— Enregistrement des notifications envoyées (*notifications*).
— Configuration personnalisée via *notification_preferences*.
— Support multicanal : email, SMS, push, in-app.

1.4 Architecture Microservices du Système de Transport Urbain

Le système de transport urbain est conçu selon une **architecture microservices**, où chaque service est autonome et possède sa propre base de données. Cette conception permet une scalabilité indépendante, une maintenance simplifiée et une résilience améliorée.

Composants principaux

— **Clients :** Web App, Mobile App, Driver App
— **API Gateway :** Load Balancer, Authentication, Rate Limiting, Routing
— **Microservices :** User Service, Ticket Service, Route Service, Geolocation Service, Subscription Service, Notification Service, Payment Service
— **Bases de données :** User DB, Ticket DB, Route DB, Location DB, Subscription DB
— **Message Broker :** RabbitMQ ou Kafka pour la communication asynchrone
— **Services externes :** Google Maps API, Payment Gateway, Email Service, SMS Service
— **Infrastructure Cloud :** Docker pour la containerisation et Kubernetes pour l'orchestration, le scaling automatique et la découverte de services

Flux et connexions

— Les clients (Web, Mobile, Driver App) communiquent avec l'API Gateway.

— L'API Gateway distribue les requêtes vers les microservices appropriés.
— Chaque microservice gère sa propre base de données.
— Les microservices publient des événements vers le Message Broker, consommés notamment par le Notification Service.
— Ticket Service et Subscription Service interagissent avec le Payment Service et la passerelle de paiement externe.
— Geolocation Service utilise Google Maps API pour la localisation et le routage.
— Notification Service envoie des notifications via Email et SMS.
— Docker et Kubernetes assurent le déploiement, la gestion des conteneurs et le scaling automatique des microservices.

Cette architecture assure la modularité, la flexibilité et la robustesse du système de transport urbain, tout en facilitant l'extension future et l'intégration de nouveaux services.

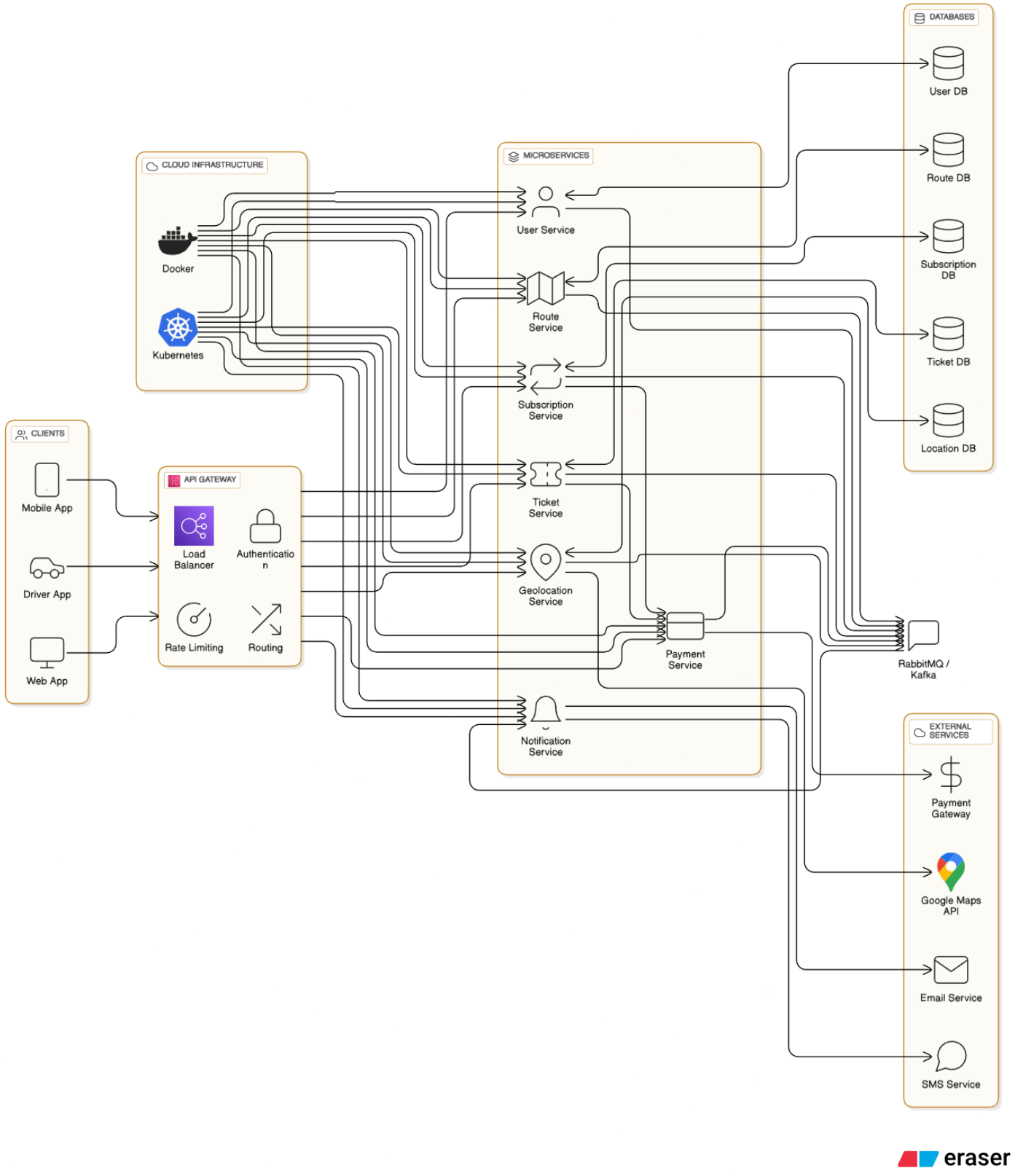

FIGURE 1.13 – Architecture Microservices