École Nationale Supérieure
d'Informatique et d'Analyse des Systèmes
- Rabat

Projet SOA/Microservices

Développement d'un Système de Gestion de Transport Urbain basé sur une Architecture Microservices

---

Élèves :
Prénom Nom
Prénom Nom
Prénom Nom

Encadrant :
Mahmoud NASSAR

Année Universitaire 2025-2026

Remerciements

Résumé

Liste des Abréviations

Introduction générale

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Contexte général du projet

1.1 Contexte du projet

Dans un environnement urbain en constante évolution, les entreprises de transport public font face à des défis croissants pour répondre aux attentes des usagers modernes. Ces derniers recherchent désormais des services digitalisés, accessibles en temps réel et offrant une expérience utilisateur fluide comparable à celle des applications mobiles du secteur privé.
Une entreprise de transport urbain souhaite se doter d'un système de gestion moderne pour améliorer ses services et l'expérience de ses usagers. Face à la concurrence des services de mobilité alternatifs (VTC, covoiturage, trottinettes électriques) et aux exigences accrues des citoyens en matière de services numériques, l'entreprise a décidé d'engager une transformation digitale ambitieuse en adoptant une architecture moderne et évolutive.
L'objectif est de créer un système, conçu dès le départ selon les meilleures pratiques architecturales, permettant d'offrir des services digitaux innovants tout en garantissant évolutivité, scalabilité et maintenabilité à long terme.

1.2 Problématique

Dans le contexte d'une mobilité urbaine de plus en plus concurrentielle, les entreprises de transport public doivent relever plusieurs défis simultanés. D'une part, elles doivent offrir des services numériques fiables, en temps réel et faciles à utiliser pour répondre aux attentes des usagers modernes. D'autre part, elles doivent rester capables d'évoluer rapidement face aux innovations technologiques et aux nouveaux usages.

Les approches architecturales traditionnelles monolithiques présentent des limitations importantes dans ce contexte. Elles freinent l'ajout de nouvelles fonctionnalités (géolocalisation temps réel, achat en ligne, notifications), compliquent la scalabilité lors des pics de charge, limitent la résilience globale du système et empêchent l'évolution indépendante des différents modules fonctionnels.

La problématique centrale de ce projet consiste donc à concevoir un système de gestion moderne qui réponde aux exigences suivantes :

- Découpler les domaines métiers (utilisateurs, tickets, trajets, abonnements, notifications) pour une meilleure modularité
- Garantir l'autonomie des équipes de développement et l'indépendance des données
- Faciliter le déploiement continu et la mise en production fréquente de nouvelles fonctionnalités
- Maintenir une qualité de service élevée en termes de performance, sécurité et disponibilité

L'adoption d'une architecture SOA/Microservices apparaît comme la réponse appropriée à ces enjeux.

1.3 Objectifs du projet

L'objectif principal de ce projet est de concevoir et développer une application moderne de gestion du transport urbain permettant de digitaliser et d'optimiser les services offerts aux usagers. La plateforme vise à simplifier les opérations de l'entreprise de transport tout en offrant une expérience fluide, sécurisée et accessible en temps réel.

Plus précisément, le système doit permettre la gestion complète des activités liées au transport urbain, incluant :

- La consultation des horaires et des trajets disponibles
- L'achat et la gestion de tickets de transport
- Le suivi en temps réel de la localisation des bus
- La gestion des abonnements (souscription, renouvellement)
- La diffusion d'informations et notifications aux usagers

Le système s'appuiera sur une architecture basée sur les microservices afin d'assurer évolutivité, performance, résilience et maintenabilité à long terme. Cette approche permettra également de répondre aux besoins de déploiement continu et d'innovation rapide, tout en garantissant l'autonomie des équipes de développement.

Chapitre 2

Analyse des besoins

2.1 Introduction

L'analyse des besoins constitue une étape fondamentale dans la conception du système de gestion de transport urbain. Elle permet d'identifier et de formaliser l'ensemble des exigences fonctionnelles et non fonctionnelles qui guideront les choix architecturaux et techniques du projet. Cette analyse s'articule autour de trois axes principaux : les besoins fonctionnels qui décrivent ce que le système doit faire, les besoins non fonctionnels qui définissent comment le système doit fonctionner, et l'identification des acteurs qui interagiront avec le système.

2.2 Besoins fonctionnels

Les besoins fonctionnels décrivent les services et fonctionnalités que le système doit offrir aux différents utilisateurs. Ils sont regroupés par domaine métier correspondant aux futurs microservices.

2.2.1 Gestion des utilisateurs

Le système doit permettre la gestion complète du cycle de vie des utilisateurs, incluant :

- **Inscription** : Permettre aux nouveaux utilisateurs de créer un compte avec leurs informations personnelles (nom, prénom, email, téléphone, mot de passe)
- **Authentification** : Assurer une connexion sécurisée via identifiants (email/mot de passe) avec gestion de session
- **Gestion de profil** : Permettre aux utilisateurs de consulter et modifier leurs informations personnelles
- **Gestion des rôles** : Différencier les types d'utilisateurs (Passager, Conducteur, Administrateur) avec des droits d'accès spécifiques
- **Récupération de mot de passe** : Offrir un mécanisme sécurisé de réinitialisation en cas d'oubli
- **Déconnexion** : Permettre la fermeture sécurisée de session

2.2.2 Gestion des tickets

Le système doit gérer l'ensemble du processus d'achat et de gestion des tickets de transport :

- **Achat de tickets** : Permettre l'achat en ligne de tickets via un processus de paiement sécurisé

- **Consultation de l'historique** : Permettre aux utilisateurs de visualiser l'historique complet de leurs achats
- **Visualisation des tickets actifs** : Afficher les tickets valides et non utilisés

2.2.3 Gestion des trajets et horaires

Le système doit fournir toutes les informations relatives aux lignes de bus et aux horaires :

- **Consultation des lignes** : Afficher la liste de toutes les lignes de bus disponibles avec leurs caractéristiques (numéro, nom, terminus)
- **Consultation des horaires** : Présenter les horaires théoriques de passage pour chaque ligne et arrêt
- **Recherche d'itinéraires** : Permettre la recherche de trajets entre deux points avec calcul des correspondances possibles

2.2.4 Géolocalisation en temps réel

Le système doit offrir un suivi en temps réel de la position des bus :

- **Localisation des bus** : Afficher la position GPS en temps réel de tous les bus en service
- **Visualisation cartographique** : Présenter les bus sur une carte interactive (intégration Google Maps ou OpenStreetMap)

2.2.5 Gestion des abonnements

Le système doit gérer le cycle de vie complet des abonnements :

- **Consultation des offres** : Présenter les différentes formules d'abonnement disponibles (mensuel, annuel) avec leurs conditions et tarifs
- **Souscription** : Permettre la souscription en ligne à un abonnement avec paiement sécurisé
- **Gestion des abonnements actifs** : Afficher les détails de l'abonnement en cours (type, date de début, date d'expiration, statut)
- **Renouvellement** : Offrir la possibilité de renouveler un abonnement manuellement ou automatiquement
- **Résiliation** : Permettre l'annulation d'un abonnement selon les conditions contractuelles
- **Historique des abonnements** : Conserver un historique complet des abonnements passés

2.2.6 Service de notifications

Le système doit assurer une communication proactive avec les utilisateurs :

- **Alertes de perturbations** : Notifier en temps réel les retards, annulations ou modifications de trajets

- **Notifications par email** : Envoyer des emails pour les événements importants (confirmation d'achat, renouvellement d'abonnement)
- **Notifications par SMS** : Envoyer des SMS pour les alertes urgentes

2.3 Besoins non fonctionnels

Les besoins non fonctionnels définissent les contraintes qualitatives et techniques que le système doit respecter. Ils sont essentiels pour garantir la qualité, la performance et la pérennité de la solution.

2.3.1 Performance

- Le système garantit des temps de réponse rapides grâce à des mécanismes de pagination et de chargement paresseux (lazy loading).
- Des requêtes de base de données optimisées et des réponses d'API allégées minimisent les temps de chargement et la surcharge du serveur.
- Les données de géolocalisation doivent être actualisées toutes les 10-15 secondes.

2.3.2 Scalabilité

- L'application suit une architecture à microservices, permettant aux services d'être déployés et mis à l'échelle indépendamment.
- Chaque service maintient son propre schéma de base de données, ce qui facilite la scalabilité horizontale et l'isolation des données.

2.3.3 Sécurité

- L'authentification est gérée via JWT (JSON Web Token) pour un contrôle d'accès sécurisé.
- Les sessions utilisateur et les appels API sont protégés contre tout accès non autorisé.
- Des autorisations basées sur les rôles (RBAC) garantissent un accès restreint selon les profils utilisateurs.
- Les données sensibles sont protégées en transit (via HTTPS/TLS) et au repos, assurant la confidentialité et l'intégrité des informations.

2.3.4 Maintenabilité

- Le système est organisé en microservices modulaires, chacun étant responsable d'un domaine métier spécifique.
- Le code source est géré par Git, avec des conventions de nommage claires et des pratiques de documentation rigoureuses.
- La séparation des responsabilités facilite le débogage, les mises à jour et l'extension des fonctionnalités.

2.3.5 Interopérabilité

- L'utilisation des conventions REST pour les API
- Spécification des contrats OpenAPI pour toutes les interfaces publiques
- Utilisation de JSON pour les échanges de données
- Capacité d'intégrer des services tiers (paiement, cartographie, SMS)

2.3.6 Portabilité et déploiement

- Tous les services doivent être packagés avec Docker
- Déploiement et gestion via Kubernetes
- Architecture compatible avec les principaux cloud providers (AWS, Azure, GCP)
- Pipeline d'intégration et déploiement continu (CI/CD)

2.3.7 Isolation et indépendance des données

- Chaque microservice possède sa propre base de données (Database per service)
- Aucun accès direct aux bases de données d'autres services
- Tout échange de données passe par les interfaces définies

2.3.8 Communication inter-services

- Utilisation de REST pour les appels directs
- Utilisation de message brokers (RabbitMQ) pour les événements
- **API Gateway** comme point d'entrée unique pour les clients externes
- Mécanisme de découverte automatique des services (**Service Discovery**)

2.4 Identification des acteurs

Le système sera utilisé par différents types d'acteurs, qu'ils soient humains ou systèmes externes. Leur identification permet de définir les cas d'usage et les interfaces nécessaires.

2.4.1 Acteurs humains

Passager

- **Rôle** : Utilisateur final du système de transport urbain
- **Responsabilités** :
  - Consulter les horaires et trajets disponibles
  - Acheter des tickets en ligne de manière sécurisée
  - Suivre la position des bus en temps réel
  - Gérer ses abonnements (souscription, renouvellement, résiliation)
  - Recevoir et consulter les notifications
  - Gérer son profil et ses préférences
- **Caractéristiques** : Peut être authentifié pour accéder à toutes les fonctionnalités, ou anonyme pour certaines consultations basiques (horaires, trajets)

Conducteur

- **Rôle** : Opérateur des bus
- **Responsabilités** :
  - Consulter son planning et les trajets assignés
  - Partager automatiquement sa position GPS
  - Signaler les incidents, retards ou problèmes techniques
  - Gérer son profil
- **Caractéristiques** : Toujours authentifié, accès limité aux fonctionnalités liées à son rôle

Administrateur

- **Rôle** : Gestionnaire et superviseur du système
- **Responsabilités** :
  - Gérer les utilisateurs (création, modification, suppression, gestion des rôles)
  - Gérer le référentiel des lignes, arrêts et horaires
  - Configurer les tarifs des tickets et abonnements
  - Assigner les trajets aux conducteurs
  - Consulter les statistiques d'utilisation et les rapports
  - Gérer les réclamations et incidents
  - Configurer les notifications système
- **Caractéristiques** : Privilèges élevés, toujours authentifié avec authentification renforcée (2FA recommandé)

2.4.2 Acteurs systèmes (externes)

Système de paiement

- **Rôle** : Service externe de traitement des transactions financières
- **Responsabilités** :
  - Traiter les paiements pour l'achat de tickets
  - Traiter les paiements pour les souscriptions d'abonnements
  - Gérer les paiements récurrents
  - Valider et sécuriser les transactions

Service de géolocalisation et cartographie

- **Rôle** : Fournisseur de données géographiques et cartographiques
- **Responsabilités** :
  - Fournir les données cartographiques pour l'affichage
  - Gérer l'affichage interactif des cartes
  - Calculer les itinéraires entre deux points
  - Estimer les temps de trajet

Service de notification externe

- **Rôle** : Plateforme d'envoi de notifications
- **Responsabilités** :
  - Envoyer des emails transactionnels et informatifs
  - Envoyer des SMS pour les alertes urgentes

2.5 Diagramme de cas d'utilisation global

Le diagramme ci-dessous présente une vue d'ensemble du système de gestion de transport urbain, en illustrant les principaux acteurs (passager, conducteur, administrateur) et leurs interactions avec les fonctionnalités clés du système, ainsi qu'avec les services externes (paiement, géolocalisation, notifications).

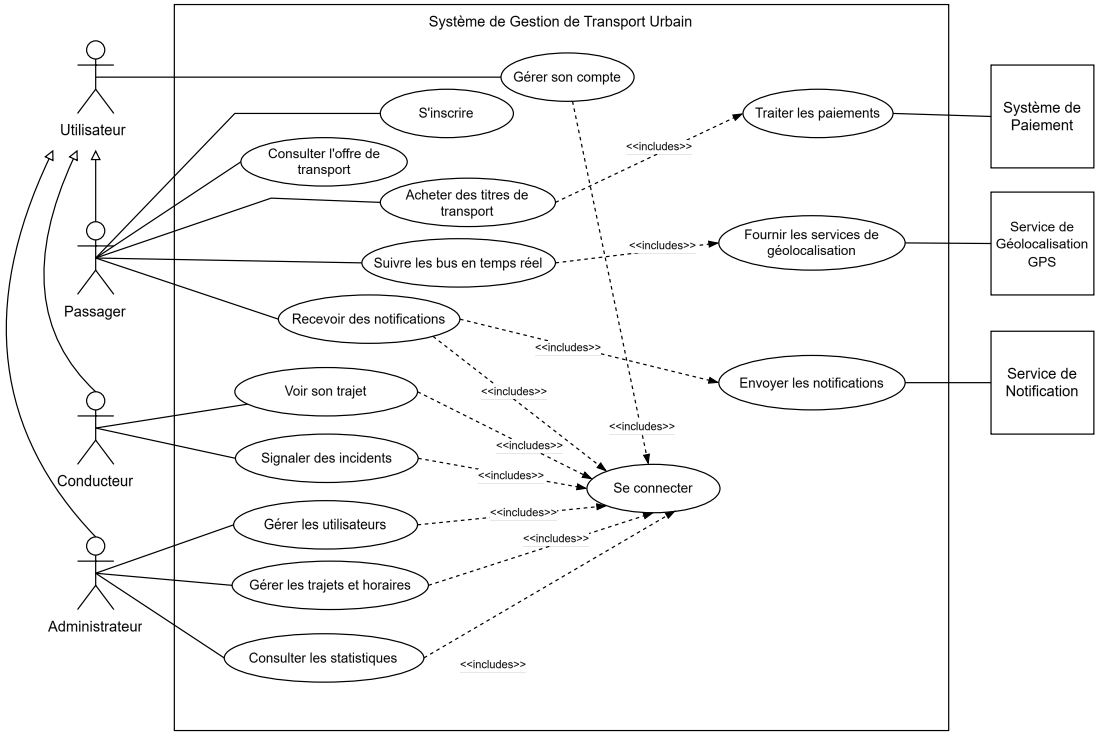

FIGURE 2.1 – Diagramme de cas d'utilisation global du système

2.6 Cas d'utilisation détaillés

Cette section présente les différents diagrammes de cas d'utilisation détaillés du système de gestion du transport urbain. Chaque diagramme illustre un sous-système fonctionnel spécifique et les interactions principales entre les acteurs et le système.

2.6.1 Gestion des utilisateurs et authentification

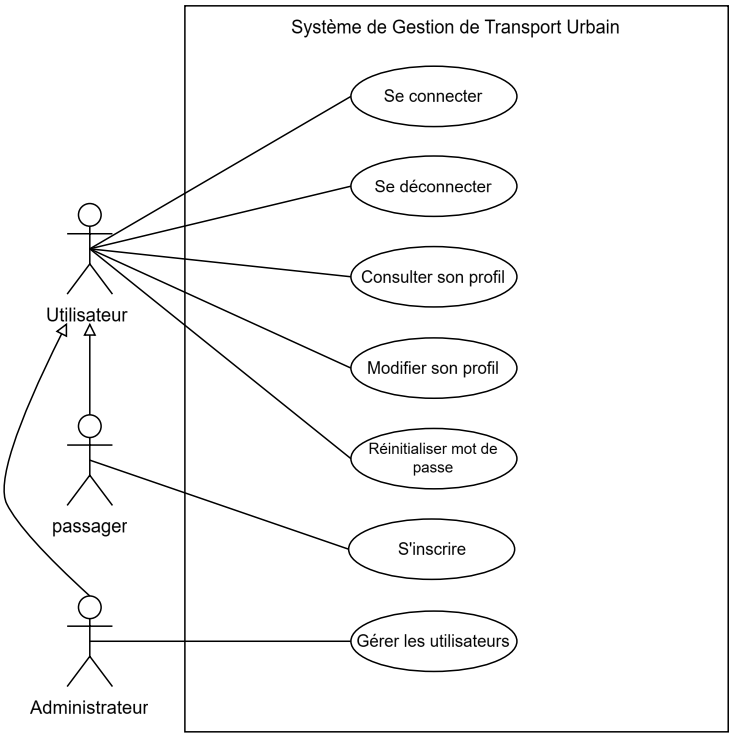

FIGURE 2.2 – Diagramme de gestion des utilisateurs et authentification

2.6.2 Diagramme de Gestion des Tickets et des Abonnements

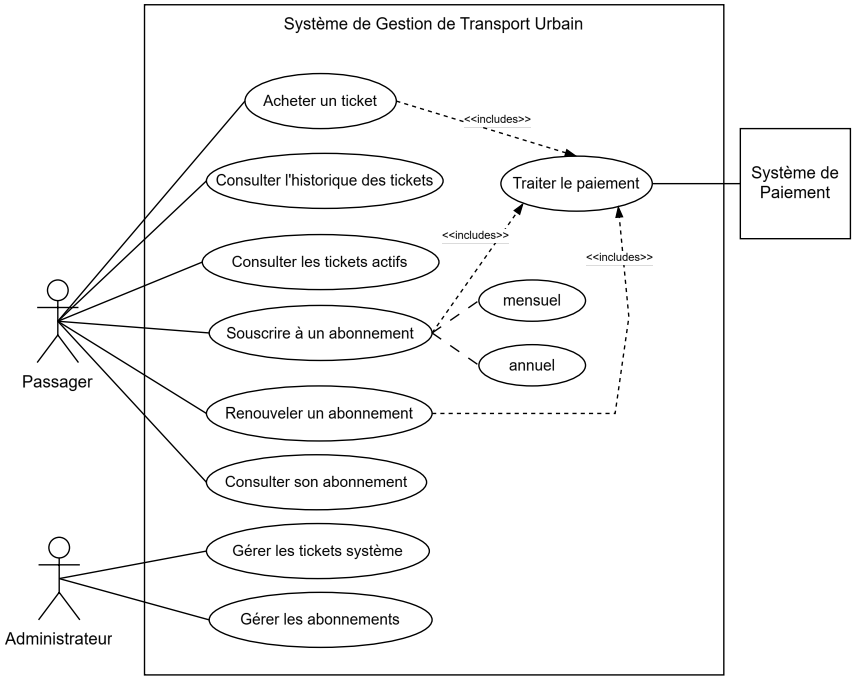

FIGURE 2.3 – Diagramme de gestion des tickets

2.6.3 Consultation des trajets et horaires

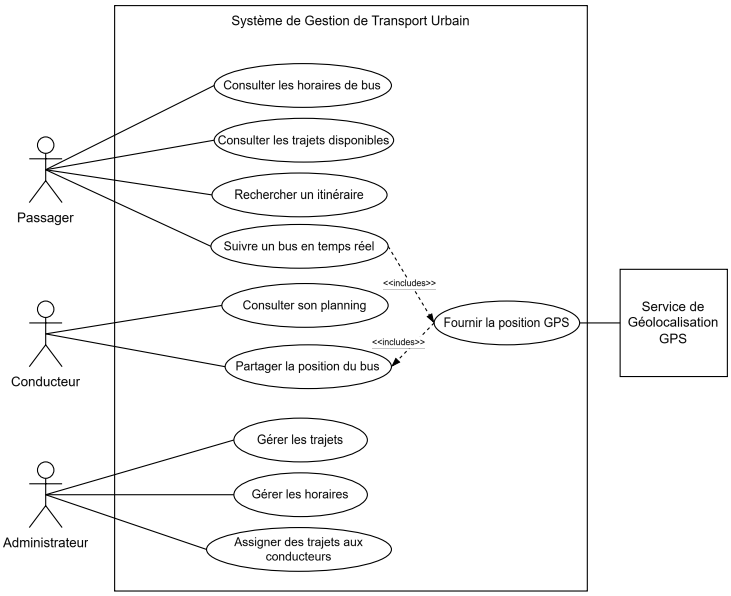

FIGURE 2.4 – Diagramme de consultation des trajets et horaires

2.6.4 Gestion des incidents et notifications

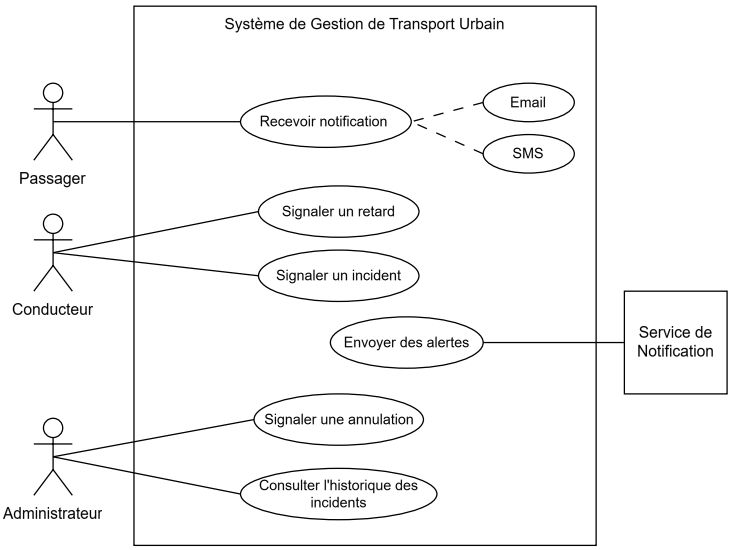

FIGURE 2.5 – Diagramme de gestion des incidents et notifications

Chapitre 3

Architecture du système

3.1 Choix architectural

3.1.1 Architecture monolithique vs microservices

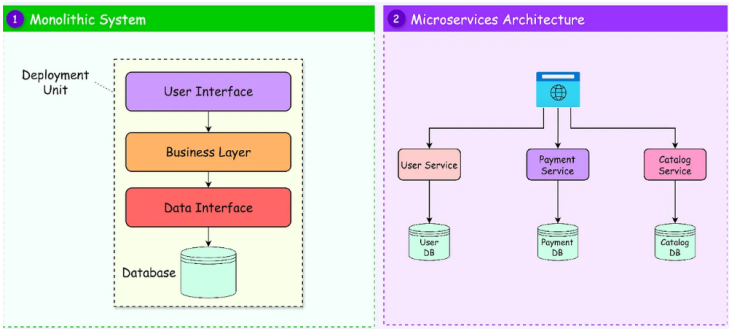

FIGURE 3.1 – Architecture monolithique vs microservices

L'architecture monolithique regroupe toutes les fonctionnalités dans une seule application, ce qui rend le déploiement simple mais limite l'évolutivité et la maintenabilité. À l'inverse, l'architecture microservices divise le système en services indépendants, chacun responsable d'un domaine métier précis et communiquant via des API légères.

3.1.2 Justification du choix

Compte tenu de la diversité des fonctionnalités (gestion des utilisateurs, tickets, abonnements, notifications, etc.), une approche microservices a été retenue. Elle permet une meilleure modularité, un déploiement indépendant de chaque service, une scalabilité horizontale et une maintenance facilitée, tout en favorisant l'intégration de nouvelles fonctionnalités.

3.1.3 Principes architecturaux adoptés

Le système repose sur les principes suivants :

- **Découplage** des services selon les domaines métiers.
- **Communication via API REST** et messages asynchrones.
- **Base de données indépendante** par service.
- **API Gateway** comme point d'accès unique.
- **Sécurité et résilience** intégrées (authentification, tolérance aux pannes).
- **Conteneurisation et orchestration** via Docker et Kubernetes.

3.2 Identification et décomposition en microservices

3.2.1 Liste des microservices identifiés

L'analyse fonctionnelle du système a permis d'identifier plusieurs microservices correspondant aux principaux domaines métiers du transport urbain. Chaque microservice est autonome, dispose de sa propre base de données et communique avec les autres via des API REST sécurisées.

- **Microservice Utilisateur** : gère l'inscription, l'authentification, la gestion des profils et des rôles (passager, conducteur, administrateur).
- **Microservice Ticket** : assure la gestion complète des titres de transport (achat, consultation, historique) et interagit avec le service de paiement.
- **Microservice Abonnement** : permet la souscription et le renouvellement des abonnements (mensuels ou annuels).
- **Microservice Trajet et Horaire** : gère les trajets, les lignes de bus, les horaires, ainsi que la recherche d'itinéraires.
- **Microservice Géolocalisation** : fournit la position GPS des bus en temps réel et permet le suivi par les passagers.
- **Microservice Notification** : envoie des alertes et messages (email ou SMS) aux utilisateurs en cas de retard, d'incident ou d'événement particulier.
- **Microservice Incident** : permet aux conducteurs de signaler des retards ou incidents et aux administrateurs de consulter leur historique.
- **Microservice Administration** : centralise les opérations de gestion du système (utilisateurs, conducteurs, statistiques).
- **Microservice Paiement** : gère le traitement sécurisé des transactions lors de l'achat de tickets ou d'abonnements.

Cette décomposition favorise la modularité, la scalabilité et la maintenabilité du système tout en facilitant le développement parallèle des différents services.

3.2.2 Architecture logique du système

Cette section présente l'organisation globale des microservices.

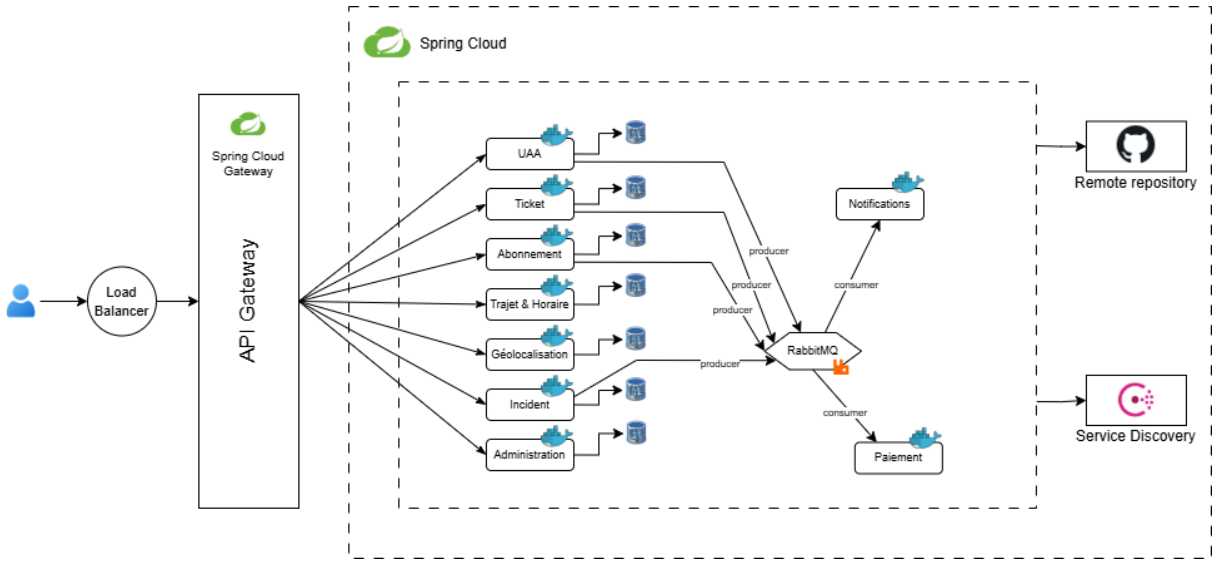

FIGURE 3.2 – Architecture du système de gestion de transport urbain

3.3 Définition des interfaces et API

3.3.1 Spécification des API par microservice

Cette section détaille les endpoints REST de chaque microservice, en précisant les méthodes HTTP, les paramètres, et les formats de requête/réponse.

API du microservice Utilisateur

Le microservice Utilisateur expose les endpoints suivants pour la gestion des comptes et l'authentification.

Endpoint	Méthode	Description
`/api/users/register`	POST	Inscription d'un nouvel utilisateur (passager)
`/api/users/login`	POST	Connexion et génération de token JWT
`/api/users/logout`	POST	Déconnexion et invalidation du token
`/api/users/profile`	GET	Consultation du profil de l'utilisateur connecté
`/api/users/profile`	PATCH	Modification du profil de l'utilisateur connecté
`/api/users/reset-password`	POST	Demande de réinitialisation du mot de passe
`/api/users/reset-password/confirm`	POST	Confirmation de la réinitialisation avec token
`/api/users`	GET	Liste de tous les utilisateurs (admin uniquement)
`/api/users/{id}`	GET	Consultation d'un utilisateur spécifique (admin)
`/api/users/{id}`	PATCH	Modification d'un utilisateur (admin)
`/api/users/{id}`	DELETE	Suppression d'un utilisateur (admin)
`/api/users/{id}/status`	PATCH	Activation/désactivation d'un compte (admin)

TABLE 3.1 – Endpoints du microservice Utilisateur

API du microservice Ticket

Conclusion Générale

Bibliographie

Annexes