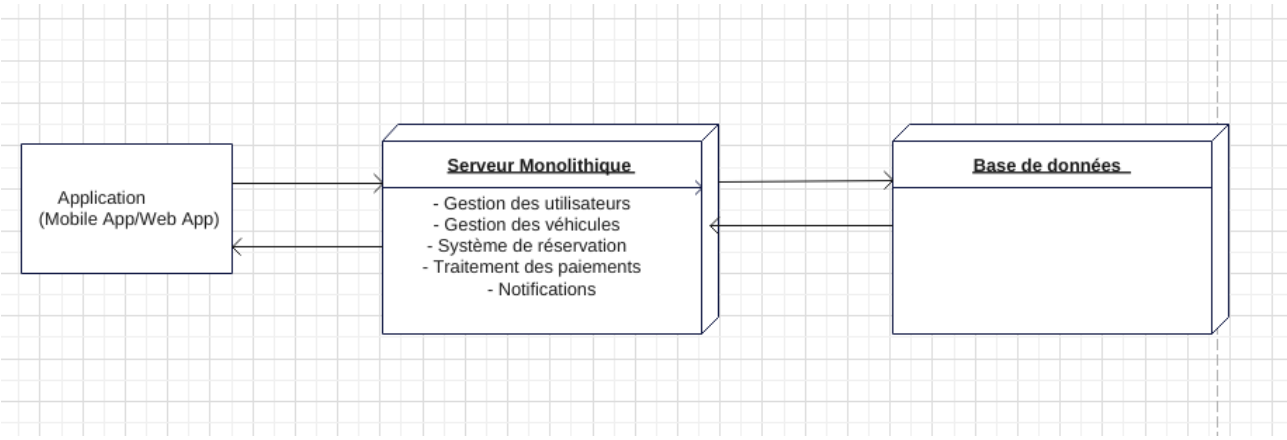
1. Représentation de l’ancienne architecture de l’application de véhicule de Propelize

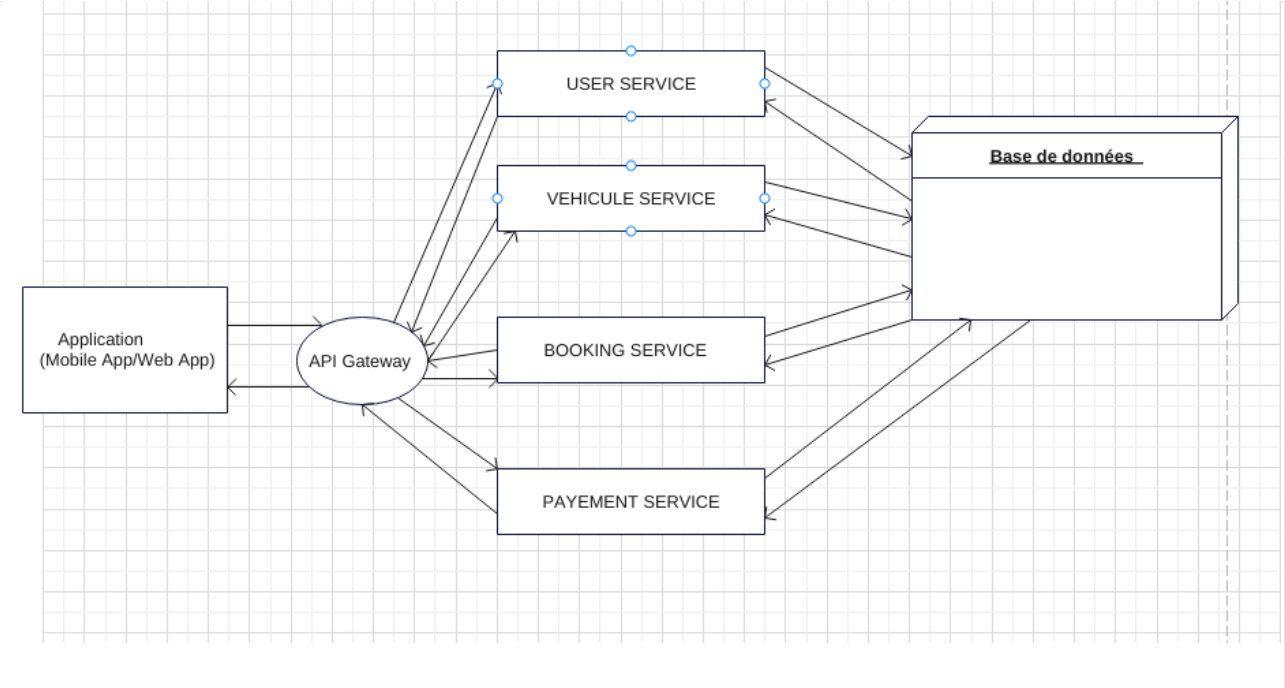


2.Inconvénients de l’architecture monolithique de Propelize

- **Complexité accrue** : À mesure que l'application grandit, la base de code devient difficile à gérer et à maintenir.
- **Cycles de déploiement longs** : Toute modification nécessite de reconstruire et redéployer l'ensemble de l'application.
- **Problèmes de fiabilité** : Un bogue dans une partie du système peut entraîner la panne de l'ensemble de l'application.
- **Difficulté d’intégration de nouvelles technologies** : La mise à jour ou l'ajout de nouvelles technologies est compliqué et nécessite souvent des réécritures majeures.
- **Maintenance difficile** : La base de code enchevêtrée rend difficile la compréhension et la modification du code, en particulier pour les nouveaux développeurs.

3. Proposition d'une architecture microservices pour l’application de location de véhicules

Voici une architecture microservices proposée pour l'application de location de véhicules de Propelize :



4. Choix d'une stack technologique pour l'implémentation du service véhicule

Stack Technologique

Langage : java

Framework : Spring boot

Base de données : Mysql

Containerisation : Docker

Orchestration : Kubernetes

Justification

Java : Adapté pour les applications I/O intensives, performant et bien supporté pour les microservices.

Spring-boot : Léger et flexible, facilite la création d'API RESTful.

Mysql : Base de données SQL adaptée pour les données flexibles et évolutives.

Docker & Kubernetes : Simplifient le déploiement, la gestion et le scaling des microservices.

6. Document de Conception de l'API de Gestion des Véhicules

Conception de l'API

Compréhension des Business Requirements (fonctionnels et non-fonctionnels)

Fonctionnels:

- Permettre la création, la mise à jour, la suppression et la recherche de véhicules.
- Fournir des fonctionnalités de recherche par immatriculation et par prix.

Non-fonctionnels:

- Performance : Réponses rapides pour assurer une bonne expérience utilisateur.
- Scalabilité : Doit pouvoir gérer une augmentation du nombre d'utilisateurs et de véhicules.
- Sécurité : Assurer la protection des données des véhicules et des utilisateurs.
- Maintenabilité : Faciliter la mise à jour et la maintenance du code.
- Identification de la cible de l’application
- L'application cible est un service de location de véhicules destiné à être utilisé par les clients de Propelize via une application web et mobile.
- Les utilisateurs finaux sont des particuliers et des entreprises cherchant à louer des voitures et des camionnettes.

Établissement des objectifs de cette application

- Offrir une expérience utilisateur fluide pour la gestion des véhicules.
- Fournir une interface API robuste et sécurisée pour les opérations CRUD (Create, Read, Update, Delete) des véhicules.
- Permettre des recherches efficaces de véhicules par immatriculation et par prix.
- Identification des Ressources et des Actions

Ressources:

- Véhicules

Actions:

- Création d'un véhicule
- Récupération des informations d'un véhicule
- Mise à jour d'un véhicule
- Suppression d'un véhicule
- Recherche de véhicules par immatriculation

- Recherche de véhicules par prix
- Le choix des Designs Principles (REST, GraphQL, SOAP)
- Design Principle Choisi: REST

REST (Representational State Transfer) est choisi pour sa simplicité, sa scalabilité et sa compatibilité avec les applications web et mobiles modernes. I

Définition des EndPoints

GET /vehicles - Récupérer la liste de tous les véhicules.

POST /vehicle - Créer un nouveau véhicule.

GET /vehicle/{id} - Récupérer les informations d'un véhicule spécifique par ID.

PUT /vehicle/{id} - Mettre à jour les informations d'un véhicule spécifique par ID.

DELETE /vehicle/{id} - Supprimer un véhicule spécifique par ID.

GET /vehicles/search/{registrationNumber } - Rechercher un véhicule par immatriculation.

GET /vehicles/price/{price} - Rechercher des véhicules par prix.

Considérations sur la sécurité

Authentification:

- Utilisation de JWT (JSON Web Tokens) pour vérifier l'identité des utilisateurs.

Authorization:

- Vérification des rôles et des permissions pour s'assurer que seules les actions autorisées sont effectuées par les utilisateurs.

Compliance and Data Protection:

- Conformité avec le RGPD pour la protection des données des utilisateurs en Europe.
- Cryptage des données sensibles en transit (TLS) et au repos.