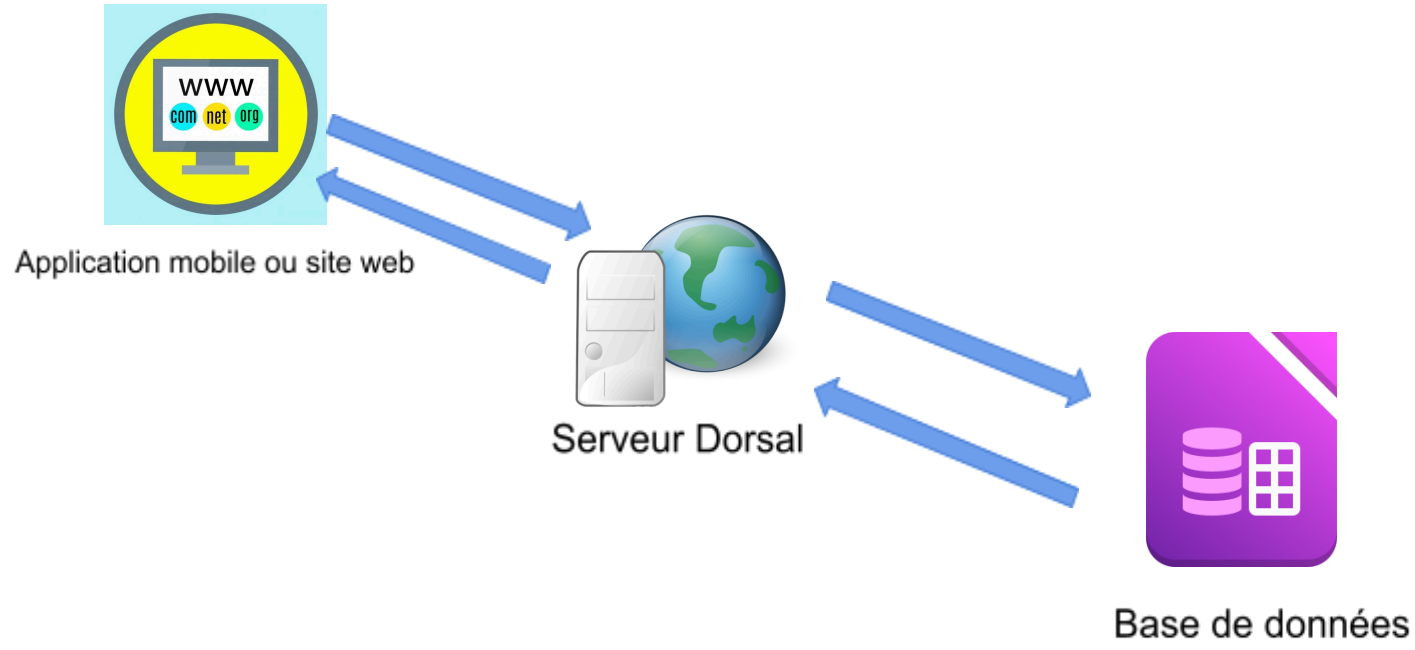
1. Architecture monolithique existante de Propelize

L'architecture actuelle de Propelize est une application monolithique traditionnelle. Cela signifie qu'il s'agit d'une seule application qui regroupe toutes les fonctionnalités de l'application de location de véhicules.



2. Inconvénients de l'architecture monolithique

- Scalabilité limitée: Il devient difficile de faire évoluer l'application horizontalement pour répondre à une demande croissante.
- Déploiement complexe: Des déploiements fréquents sont difficiles car l'ensemble de l'application doit être déployé en même temps.
- Mauvaise maintenabilité: La base de code monolithique devient complexe et difficile à gérer à mesure qu'elle grandit.
- Fiabilité réduite: Un bogue ou une défaillance dans une partie du système peut affecter l'ensemble de l'application.
- Flexibilité limitée: Il est difficile d'adopter de nouvelles technologies ou de modifier des fonctionnalités spécifiques.

3. Architecture de microservices proposée

Pour remédier aux limitations de l'architecture monolithique, Propelize peut adopter une architecture de microservices. Dans cette approche, l'application est décomposée en services indépendants et modulaires, chacun avec sa propre responsabilité bien définie. Les avantages incluent:

- Scalabilité améliorée: Les microservices peuvent être mis à l'échelle indépendamment en fonction de la demande.
- Déploiements plus simples: Des déploiements fréquents sont possibles pour des services spécifiques sans affecter l'ensemble de l'application.
- Meilleure maintenabilité: Les microservices plus petits sont plus faciles à comprendre et à maintenir.
- Fiabilité accrue: La défaillance d'un service n'affecte pas l'ensemble de l'application.
- Flexibilité accrue: Il est plus facile d'adopter de nouvelles technologies ou de modifier des fonctionnalités spécifiques dans des microservices individuels.

4. Choix de la pile technologique pour le service de gestion des véhicules

Pour implémenter le service de gestion des véhicules, une pile technologique composée de:

- Langage de programmation: JavaScript car il est beaucoup répandu et son implémentation est très simple et surtout une gamme de framework basé sur lui pour nous rendre la tâche encore plus facile
- Framework de développement web: nodejs
- Base de données: fichier json nous faison le choix d'un fichier json pour que le test puisse s'effectuer dans n'importe quel environnement sans devoir installer de modules complémentaire pour une base de données
- Conteneurisation: Docker (standard de l'industrie pour la conteneurisation des applications)

5. API de gestion des véhicules

L'API de gestion des véhicules expose des points de terminaison pour gérer les informations sur les véhicules, y compris(Test effectuer avec postman)

GET /vehicles: Récupérer tous les véhicules
GET /vehicles/{id}: Récupérer un véhicule spécifique par ID
POST /vehicles: Créer un nouveau véhicule
PUT /vehicles/{id}: Mettre à jour un véhicule existant
DELETE /vehicles/{id}: Supprimer un véhicule
GET /vehicles/search/RegistrationNumber
GET /vehicles/search/rentalPrice

6. Documentation du processus

le processus suit les étapes suivantes

- Conception de l'API
- Justification de la pile technologique
- Implémentation de l'API
- Intégration de la base de données
- Déploiement avec Docker

7. Utilisation d'une base de données

nous allons utiliser un fichier Json pour permettre son test sans l’utilisation de module supplémentaire pour la base de données lors du test

8. Déploiement avec Docker Compose

Docker Compose sera utilisé pour simplifier le déploiement et la gestion de l'application de microservices. Un fichier docker-compose.yml sera créé pour définir les services Docker pour l'API et la base de données, et les commandes docker-compose up et docker-compose down seront utilisées pour démarrer et arrêter l'application.